Apprendre le Javascript

www.ccim.be/ccim328/js/index.htm

Bref préambule

Partagés entre le copier/coller de Javascripts glanés de gauche à droite sur le Web, furieux de voir échouer les quelques modifications apportées, dépités devant la documentation à priori hermétique de Netscape et un peu "nuls" en programmation, vous souhaitez peut-être comme moi comprendre un peu plus ce langage qui met un peu de piment dans les pages Html. Ce tutorial vous est destiné.

L'apprentissage d'un langage de programmation, fut-il aussi simpliste que Javascript (c'est pourtant bien ce que prétendent certains!!!), implique la connaissance d'une nébuleuse d'éléments avant de pouvoir mettre en oeuvre ceux-ci. Pour des raisons pédagogiques, nous avons conçu ce tutorial pour une lecture à deux niveaux :

- *un niveau débutant* qui rassemble les notions de base de Javascript.
- *un niveau avancé* (noté +) pour aller un peu plus loin dans ces concepts (sans prétendre cependant à l'expertise).

L'auteur vous souhaite un apprentissage fructueux de Javascript.

Chapitre 1 : Javascript

Javascript est un langage de scripts
qui incorporé aux balises Html,
permet d'améliorer la présentation
et l'interactivité des pages Web.

Javascript est donc une extension du code Html des pages Web. Les scripts, *qui s'ajoutent ici aux balises Html*, peuvent en quelque sorte être comparés aux macros d'un traitement de texte.

Ces scripts vont être gérés et exécutés par le browser lui-même sans devoir faire appel aux ressources du serveur. Ces instructions seront donc traitées en direct et surtout sans retard par le navigateur.

Javascript a été initialement développé par Netscape et s'appelait alors LiveScript. Adopté à la fin de l'année 1995, par la firme Sun (qui a aussi développé Java), il prit alors son nom de Javascript.

Javascript n'est donc pas propre aux navigateurs de Netscape (bien que cette firme en soit un fervent défenseur). Microsoft l'a d'ailleurs aussi adopté à partir de son Internet Explorer 3. On le retrouve, de façon améliorée, dans Explorer 4.

Les versions de Javascript se sont succédées avec les différentes versions de Netscape : Javascript pour Netscape 2, Javascript 1.1 pour Netscape 3 et Javascript 1.2 pour Netscape 4. Ce qui n'est pas sans poser certains problèmes de compatibilité, selon le browser utilisé, des pages comportant du code Javascript. Mais consolons nous en constatant qu'avec MSIE 3.0 ou 4.0 et la famille Netscape, une très large majorité d'internautes pourra lire les pages comprenant du Javascript.

L'avenir de Javascript est entre les mains des deux grands navigateurs du Web et en partie lié à la guerre que se livrent Microsoft et Netscape. On s'accorde à prédire un avenir prometteur à ce langage surtout de par son indépendance vis à vis des ressources du serveur.

Chapitre 2 : Javascript n'est pas Java

Il importe de savoir que Javascript est totalement différent de Java. Bien que les deux soient utilisés pour créer des pages Web évoluées, bien que les deux reprennent le terme Java (café en américain), nous avons là deux outils informatiques bien différents.

Javascript	**Java**
Code intégré dans la page Html	Module (applet) distinct de la page Html
Code interprété par le browser au moment de l'exécution	Code source compilé avant son exécution
Codes de programmation simples mais pour des applications limitées	Langage de programmation beaucoup plus complexe mais plus performant
Permet d'accéder aux objets du navigateur	N'accède pas aux objets du navigateur
Confidentialité des codes nulle (code source visible)	Sécurité (code source compilé)

Plus simplement :

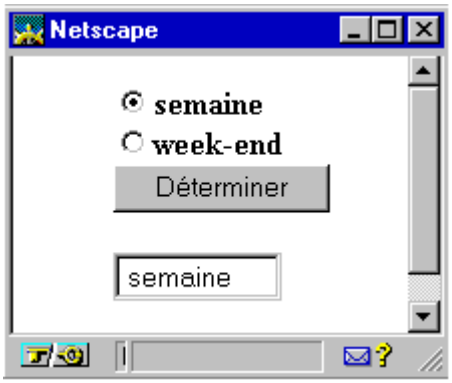
- Javascript est plus simple à mettre en oeuvre car c'est du code que vous ajouterez à votre page écrite en Html avec par exemple un simple éditeur de texte comme Notepad. Java pour sa part, nécessite une compilation préalable de votre code.
- Le champ d'application de Javascript est somme toute assez limité alors qu'en Java vous pourrez en principe tout faire.
- Comme votre code Javascript est inclus dans votre page Html, celui-ci est visible et peut être copié par tout le monde (view source). Ce qui pour les entreprises (et les paranoïaques) est assez pénalisant. Par contre, en Java, votre code source est broyé par le compilateur et est ainsi indéchiffrable.
- Même si c'est une appréciation personnelle, les codes Javascript ne ralentissent pas le chargement de la page alors que l'appel à une applet Java peut demander quelques minutes de patience supplémentaire à votre lecteur.

Chapitre 3 : Un peu de théorie objet

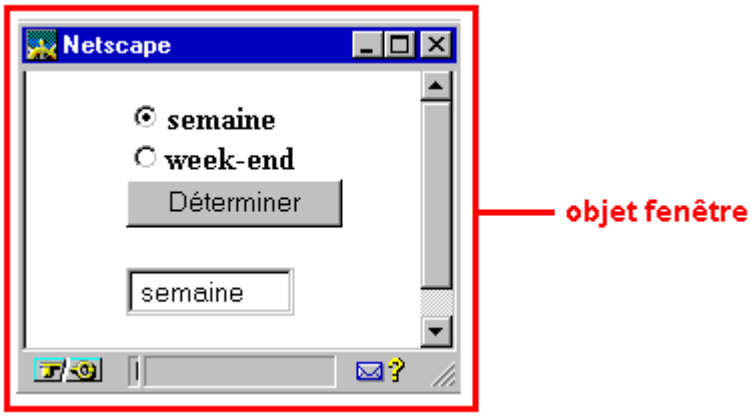
3.1 Les objets et leur hiérarchie

En bon internaute, vous voyez sur votre écran une page Web.
Javascript va diviser cette page en objets et surtout va vous permettre d'accéder à ces objets, d'en retirer des informations et de les manipuler.

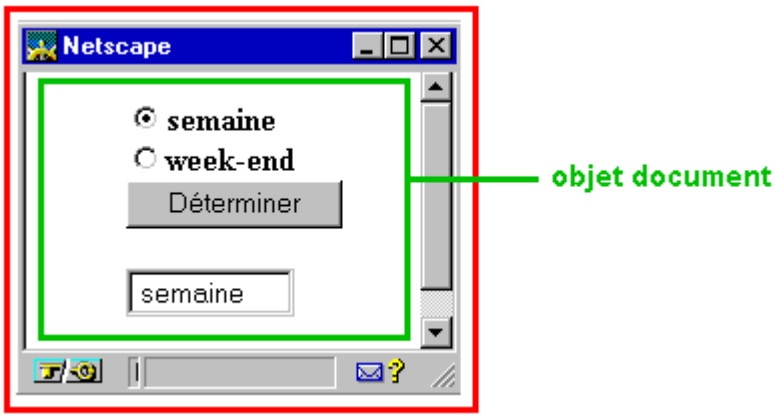
Voyons d'abord une illustration des différents objets qu'une page peut contenir.
Vous avez chargé la page suivante :



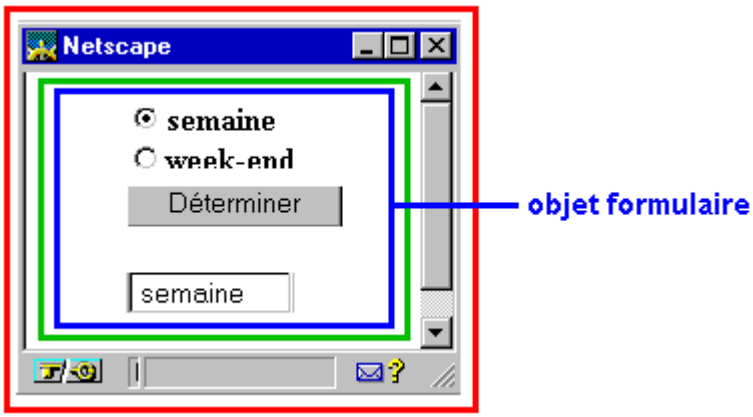
Cette page s'affiche dans une fenêtre. C'est l'objet fenêtre.



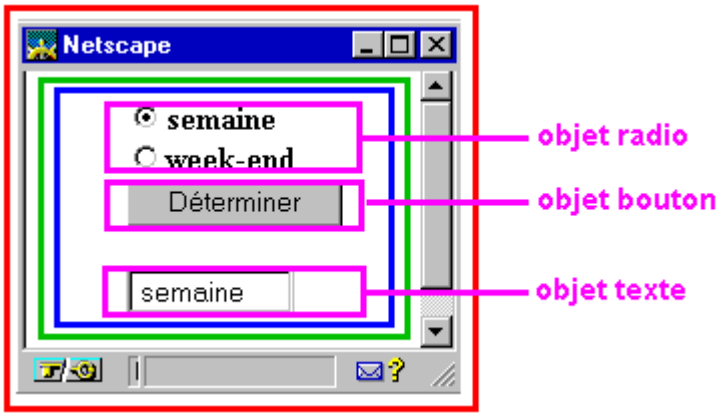
Dans cette fenêtre, il y a un document Html. C'est l'objet document. Autrement dit (et c'est là que l'on voit apparaître la notion de la hiérarchie des objets Javacript), l'objet fenêtre contient l'objet document.



Dans ce document, on trouve un formulaire au sens Html. C'est l'objet formulaire. Autrement dit, l'objet fenêtre contient un objet document qui lui contient un objet formulaire.



Dans ce document, on trouve trois objets. Des boutons radio, un bouton classique et une zone de texte. Ce sont respectivement l'objet radio, l'objet bouton, l'objet texte. Autrement dit l'objet fenêtre contient l'objet document qui contient l'objet formulaire qui contient à son tour l'objet radio, l'objet fenêtre contient l'objet document qui contient l'objet formulaire qui contient à son tour l'objet bouton et l'objet fenêtre contient l'objet document qui contient l'objet formulaire qui contient à son tour l'objet texte.



La hiérarchie des objets de cet exemple est donc

fenêtre document formulaire radio bouton texte

Pour accéder à un objet (vous l'avez peut-être déjà deviné), il faudra donner le chemin complet de l'objet en allant du contenant le plus extérieur à l'objet à l'objet référencé.
Soit par exemple pour le bouton radio "semaine" : (window).document.form.radio[0].
Nous avons mis l'objet window entre parenthèses car comme il occupe la première place dans la hiérarchie, il est repris par défaut par Javascript et devient donc facultatif.

Et enfin pour les puristes, Javascript n'est pas à proprement parler un langage orienté objet tel que C++ ou Java. On dira plutôt que Javascript est un langage basé sur les objets.

3.2 Les propriétés des objets

Une propriété est un attribut, une caractéristique, une description de l'objet. Par exemple, l'objet volant d'une voiture a comme propriétés qu'il peut être en bois ou en cuir. L'objet livre a comme propriétés son auteur, sa maison d'édition, son titre, son numéro ISBN, etc.

De même les objets Javascript ont des propriétés personnalisées. Dans le cas des boutons radio, une de ses propriétés est, par exemple, sa sélection ou sa non-sélection (checked en anglais).

En Javascript, pour accéder aux propriétés, on utilise la syntaxe :

```
nom_de_l'objet.nom_de_la_propriété
```

Dans le cas du bouton radio "semaine", pour tester la propriété de sélection, on écrira

```
document.form.radio[0].checked
```

Chapitre 4 : Vos outils pour le Javascript

Pour apprendre et exploiter le Javascript, il vous faut :

1. un browser qui reconnaît le Javascript.
2. une solide connaissance du Html
3. un simple éditeur de texte

4.1 Un browser compatible Javascript

Uniquement Netscape et Microsoft vous proposent des navigateurs Javascript "enabled". Pour Microsoft à partir de MSIE Explorer 3.0 et Netscape à partir de Netscape Navigator 2.0.

Par contre, il faut être attentif aux versions de Javascript exploitées par ces browers.

Netscape 2.0	Javascript (baptisé à posteriori 1.0)
Netscape 3.0	Javascript 1.1
Netscape 4.0 (Communicator)	Javascript 1.2

Explorer 3.0	Quelque chose qui ressemble à du Javascript 1.0
Explorer 4.0	Javascript 1.2

Il faut bien admettre que Javascript est plutôt l'affaire de Netscape et que vous courrez au devant d'une collection d'ennuis en utilisant Explorer 3 pour le Javascript.

4.2 Un solide bagage en Html

Comme le code du Javascript vient s'ajouter au "code" du langage Html, une connaissance approfondie des balises ou tags Html est souhaitable sinon indispensable. Ainsi les utilisateurs d'éditeurs Html "whsiwyg" ou autres "publishers" Html risquent de devoir retourner à leurs chères études.

Je ne peux que vous recommander un tutorial du langage Html du même auteur. ""Apprendre le langage Html" à l'adresse www.ccim.be/ccim328/html/index.htm

4.3 Un bon éditeur de texte

Une page Html n'est que du texte. Le code Javascript n'est lui aussi que du texte. Quoi de plus simple qu'un éditeur de ... texte comme le Notepad de Windows pour inclure votre Javascript dans votre page Html. Un éditeur Html de la première génération (un bon vieil éditeur qui fait encore apparaître les balises), comme HTML Notepad, fait également bien l'affaire.

De plus en plus d'éditeurs Html whsiwyg proposent une fenêtre Javascript. Attention ! Si certains semblent bien faits comme WebExpert 2 (en français) avec d'autres, il arrive que le code Javascript introduit soit modifié par l'éditeur comme FrontPage ou Netscape Gold. A vos expériences...

Ajoutons que l'on commence à voir des programmes "Visual Javascript" mais ils me semblent très lourds à gérer pour n'ajouter finalement que quelques lignes. Affaire à suivre...

Chapitre 5 : Le Javascript minimum

5.1 La balise <SCRIPT>

De ce qui précède, vous savez déjà que votre script vient s'ajouter à votre page Web.
Le langage Html utilise des tags ou balises pour "dire" au browser d'afficher une portion de texte en gras, en italique, etc.
Dans la logique du langage Html, il faut donc signaler au browser par une balise, que ce qui suit est un script et que c'est du Javascript (et non du VBScript). C'est la balise
<SCRIPT LANGUAGE="Javascript">.
De même, il faudra informer le browser de la fin du script.
C'est la balise </SCRIPT>.

5.2 Les commentaires

Il vous sera peut-être utile d'inclure des commentaires personnels dans vos codes Javascript. C'est même vivement recommandé comme pour tous les langages de programmation (mais qui le fait vraiment ?).

Javascript utilise les conventions utilisées en C et C++ soit
// commentaire
Tout ce qui est écrit entre le // et la fin de la ligne sera ignoré.

Il sera aussi possible d'inclure des commentaires sur plusieurs lignes avec le code
/* commentaire sur
plusieurs lignes */

Ne confondez pas les commentaires Javascript et les commentaires Html
(pour rappel <!-- ...-->).

5.3 Masquer le script pour les anciens browsers

Les browsers qui ne comprennent pas le Javascript (et il y en a encore) ignorent la balise <script> et vont essayer d'afficher le code du script sans pouvoir l'exécuter. Pour éviter l'affichage peu esthétique de ses inscriptions cabalistiques, on utilisera les balises de commentaire du langage Html <!-- ... -->.

Votre premier Javasript ressemblera à ceci :
<SCRIPT LANGUAGE="javascript">
<!-- Masquer le script pour les anciens browsers
...
programme Javascript
...
// Cesser de masquer le script -->
</SCRIPT>

5.4 Où inclure le code en Javascript ?

Le principe est simple. Il suffit de respecter les deux principes suivants :
- n'importe où.
- mais là où il le faut.

Le browser traite votre page Html de haut en bas (y compris vos ajoutes en Javascript). Par conséquent, toute instruction ne pourra être exécutée que si le browser possède à ce moment précis tous les éléments nécessaires à son exécution. Ceux-ci doivent donc être déclarés avant ou au plus tard lors de l'instruction.

Pour s'assurer que le programme script est chargé dans la page et prêt à fonctionner à toute intervention de votre visiteur (il y a des impatients) on prendra l'habitude de déclarer systématiquement (lorsque cela sera possible) un maximum d'éléments dans les balises d'en-tête soit entre <HEAD> et </HEAD> et avant la balise <BODY>. Ce sera le cas par exemple pour les fonctions.
Rien n'interdit de mettre plusieurs scripts dans une même page Html.

Il faut noter que l'usage de la balise script n'est pas toujours obligatoire. Ce sera le cas des événements Javascript (par exemple onClick) où il faut simplement insérer le code à l'intérieur de la commande Html comme un attribut de celle-ci. L'événement fera appel à la fonction Javascript lorsque la commande Html sera activée. Javascript fonctionne alors en quelque sorte comme une extension du langage Html.

5.5 Une première instruction Javascript

Sans vraiment entrer dans les détails, voyons une première instruction Javascript (en fait une méthode de l'objet window) soit l'instruction alert().

```
alert("votre texte");
```

Cette instruction affiche un message (dans le cas présent votre texte entre les guillemets) dans une boite de dialogue pourvue d'un bouton OK. Pour continuer dans la page, le lecteur devra cliquer ce bouton.

Vous remarquerez des points-virgules à la fin de chaque instruction Javascript (ce qui n'est pas sans rappeler le C et le C++). Le Javascript, bon enfant, est moins strict que ces autres langages et ne signale généralement pas de message d'erreur s'ils venaient à manquer. On peut considérer que le point-virgule est optionnel et qu'il n'est obligatoire que lorsque vous écrivez plusieurs instructions sur une même ligne. On recommande quand même vivement dans la littérature d'en mettre de façon systématique.

Javascript est "bon enfant" car il n'est pas toujours trop strict sur la syntaxe et passe au-dessus de certaines libertés prises avec celle-ci. Très bien! Mais ce caractère "bon enfant" est à double tranchant car parfois, pour une raison indéterminée, il devient dans certaines situations plus rigoureux et alors bonne chance pour débugger votre script.

5.6 Votre première page Html avec du Javascript

<HTML>	Html normal
<HEAD>	...
<TITLE>Mon premier Javascript</TITLE>	...
</HEAD>	...
<BODY>	...
Bla-bla en Html	...
<SCRIPT LANGUAGE="Javascript">	Début du script
<!--	Masquer le script
alert("votre texte");	Script
//-->	Fin de masquer
</SCRIPT>	Fin du script
Suite bla-bla en Html	Html normal
</BODY>	...
</HTML>	...

5.7 Remarques

Javascript est case sensitive. Ainsi il faudra écrire alert() et non Alert(). Pour l'écriture des instructions Javascript, on utilisera l'alphabet ASCII classique (à 128 caractères) comme en Html. Les caractères accentués comme é ou à ne peuvent être employés que dans les chaînes de caractères c.-à-d. dans votre texte de notre exemple.

Les guillemets " et l'apostrophe ' font partie intégrante du langage Javascript. On peut utiliser l'une ou l'autre forme à condition de ne pas les mélanger. Ainsi alert("...') donnera un message d'erreur. Si vous souhaiter utiliser des guillemets dans vos chaînes de caractères, tapez \" ou \' pour les différencier vis à vis du compilateur.

+5.8 Versions du langage Javascript

Avec les différentes versions déjà existantes (Javascript 1.0, Javascript 1.1 et Javascript 1.2), on peut imaginer des scripts adaptés aux différentes versions mais surtout aux différents navigateurs ;

```
<SCRIPT LANGUAGE="Javascript">
// programme pour Netscape 2 et Explorer 3
var version="1.0";
</SCRIPT>

<SCRIPT LANGUAGE="Javascript1.1">
// programme pour Netcape 3 et Explorer 4
var version=1.1;
</SCRIPT>

<SCRIPT LANGUAGE="Javascript1.2">
// programme pour Netscape 4
var version=1.2;
</SCRIPT>

<SCRIPT LANGUAGE="Javascript">
document.write('Votre browser supporte le Javascript ' + version);
</SCRIPT>
```

+5.9 Extension .js pour scripts externes

Il est possible d'utiliser des fichiers externes pour les programmes Javascript. On peut ainsi stocker les scripts dans des fichiers distincts (avec l'extension .js) et les appeler à partir d'un fichier Html. Le concepteur peut de cette manière se constituer une bibliothèque de script et les appeler à la manière des #include du C ou C++. La balise devient

<SCRIPT LANGUAGE='javascript' SRC='http://site.com/javascript.js'></SCRIPT>

+5.10 Toujours des commentaires

Outre les annotations personnelles, les commentaires peuvent vous être d'une utilité certaine en phase de débuggage d'un script pour isoler (sans effacer) une ligne suspecte.

Pour les esprits compliqués, notons que les commentaires ne peuvent être imbriqués sous peine de message d'erreur. La formulation suivante est donc à éviter :
/* script réalisé ce jour /* jour mois */
et testé par nos soins*/

+5.11 Alert() ... rouge

Joujou des débutants en Javascript, cette petite fenêtre est à utiliser avec parcimonie pour attirer l'attention du lecteur pour des choses vraiment importantes. Et puis, elles ne sont vraiment pas destinées à raconter sa vie. Javascript met à votre disposition la possibilité de créer de nouvelles fenêtres de la dimension de votre choix qui apparaissent un peu comme les popup des fichiers d'aide. Nous les étudierons plus loin dans l'objet Window.

Alert() est une méthode de l'objet Window. Pour se conformer à la notation classique nom_de_l'objet.nom_de_la_propriété, on aurait pu noter window.alert(). Window venant en tête des objets Javascript, celui-ci est repris par défaut par l'interpréteur et devient en quelque sorte facultatif.

Si vous souhaitez que votre texte de la fenêtre alert() s'inscrive sur plusieurs lignes, il faudra utiliser le caractère spécial /n pour créer une nouvelle ligne.

Chapitre 6 : Afficher du texte

6.1 Méthode de l'objet document

Rappelez-vous... Nous avions montré que ce qui apparaît sur votre écran, peut être "découpé" en objets et que Javascript allait vous donner la possibilité d'accéder à ces objets (Un peu de théorie objet). La page Html qui s'affiche dans la fenêtre du browser est un objet de type document.

A chaque objet Javascript, le concepteur du langage a prévu un ensemble de méthodes (ou fonctions dédiées à cet objet) qui lui sont propres. A la méthode document, Javascript a dédié la méthode "écrire dans le document", c'est la méthode write().

L'appel de la méthode se fait selon la notation :
nom_de_l'objet.nom_de_la_méthode

Pour appeler la méthode write() du document, on notera
document.write();

6.2 La méthode write()

La syntaxe est assez simple soit
write("votre texte");

On peut aussi écrire une variable, soit la variable resultat,

write(resultat);

Pour associer du texte (chaînes de caractères) et des variables, on utilise l'instruction write("Le résultat est " + resultat);

On peut utiliser les balises Html pour agrémenter ce texte
write("<B>Le résultat est</B>" + resultat); ou
write ("<B>" + "Le résultat est " + "</B>" + resultat)

6.3 Exemple (classique !)

On va écrire du texte en Html et en Javascript.

```
<HTML>
<BODY>
<H1>Ceci est du Html</H1>
<SCRIPT LANGUAGE="Javascript">
<!--
document.write("<H1>Et ceci du Javascript</H1>");
//-->
</SCRIPT>
</BODY>
</HTML>
```

Ce qui donnera comme résultat :

Ceci est du Html
Et ceci du Javascript

+ 6.4 L'instruction writeln()

La méthode writeln() est fort proche de write() à ceci près qu'elle ajoute un retour chariot à la fin des caractères affichés par l'instruction. Ce qui n'a aucun effet en Html. Pour faire fonctionner write() Il faut l'inclure dans des balises <PRE>.

```
<PRE>
<SCRIPT LANGUAGE="Javascript">
<--
document.writeln("Ligne 1");
document.writeln("Ligne 2");
//-->
</SCRIPT>
</PRE>
```

Autrement dit l'emploi de writeln() est anecdotique et on utilise simplement le tag
 avec la méthode write().

+6.5 De la belle écriture en Javascript...

6.5.1 variable.big();

L'emploi de .big() affichera la variable comme si elle était comprise entre les balises Html <BIG></BIG>.
Les quatre instructions Javascript suivantes sont équivalentes :

```
str="Something"; (str est une variable)
document.write("<BIG>"+str+"</BIG>");
document.write('<BIG>Something</BIG>');
document.write(str.big());
```

```
document.write("Something".big());
```

6.5.2 variable.small();

L'emploi de .small() affichera la variable comme si elle était comprise entre les balises Html <SMALL> </SMALL>.
Les quatre instructions Javascript suivantes sont équivalentes :

```
str="Something";
document.write("<SMALL>"+str +"</SMALL>");
document.write("<SMALL>Something" +"</SMALL>");
document.write(str.small());
document.write("Something".small());
```

6.5.3 variable.blink();

L'emploi de .blink() affichera la variable comme si elle était comprise entre les balises Html <BLINK></BLINK>. Pour rappel, cette balise (qui est par ailleurs vite ennuyeuse) n'est valable que sous Netscape 3 et plus.
Les quatre instructions Javascript suivantes sont équivalentes :

```
str="Something";
document.write('<BLINK>'+str+'</BLINK>');
document.write("<BLINK>Something</BLINK>");
document.write(str.blink());
document.write("Something".blink());
```

6.5.4 variable.bold();

L'emploi de .bold() affichera la variable comme si elle était comprise entre les balises Html <B></B>.
Les quatre instructions Javascript suivantes sont équivalentes :

```
str="Some words";
document.write("<B>"+str+"</B>");
document.write("<B>Some words</B>");
document.write(str.bold());
document.write("Some words".bold());
```

6.5.5 variable.fixed();

L'emploi de .fixed() affichera la variable comme si elle était comprise entre les balises Html <TT></TT>.
Les quatre instructions Javascript suivantes sont équivalentes :

```
str="Something";
document.write("<TT>"+str+"</TT>");
document.write("<TT>Something</TT>");
document.write(str.fixed());
document.write("Something".fixed());
```

6.5.6 variable.italics();

L'emploi de .italics() affichera la variable comme si elle était comprise entre les balises Html <I></I>.
Les quatre instructions Javascript suivantes sont équivalentes :

```
str="Something";
document.write("<I>"+str+"</I>");
document.write("<I>Something</I>");
document.write(str.italics());
```

```
document.write("Some word".italics());
```

6.5.7 variable.fontcolor(color);

L'emploi de .fontcolor(color) affichera la variable comme si elle était comprise entre les balises Html <FONT COLOR="color"> </FONT>.
Les quatre instructions Javascript suivantes sont équivalentes :

```
str1="Some words";
str2="red";
document.write("<FONT COLOR='red'>" +str1+"</FONT>");
document.write("<FONT COLOR='red'>" +"Something</FONT>");
document.write(str1.fontcolor(str2));
document.write(str1.fontcolor("red"));
```

6.5.8 variable.fontsize(x);

L'emploi de .fontsize(x) affichera la variable comme si elle était comprise entre les balises Html <FONT SIZE="x"></FONT> où x est un nombre de 1 à 7 ou exprimé en plus ou en moins par rapport à 0 par exemple -2, -1, +1, +2.
Les quatre instructions Javascript suivantes sont équivalentes :

```
str="Something";
x=3;
document.write("<FONT SIZE=3>" +str+"</FONT>");
document.write("<FONT SIZE=3>" +"Something</FONT>");
document.write(str.fontsize(3));
document.write(str.fontsize(x));
```

6.5.9 variable.strike();

L'emploi de .strike() affichera la variable comme si elle était comprise entre les balises Html <STRIKE></STRIKE>.
Les quatre instructions Javascript suivantes sont équivalentes :

```
str="Something";
document.write("<STRIKE>"+str +"</STRIKE>");
document.write("<STRIKE>Something" +"</STRIKE>");
document.write(str.strike());
document.write("Something".strike());
```

6.5.10 variable.sub();

L'emploi de .sub() affichera la variable comme si elle était comprise entre les balises Html .
Les quatre instructions Javascript suivantes sont équivalentes :

```
str="Something";
document.write("<SUB>"+str+"</SUB>");
document.write("<SUB>Something" +"</SUB>");
document.write(str.sub());
document.write("Something".sub());
```

6.5.11 variable.sup();

L'emploi de .sup() affichera la variable comme si elle était comprise entre les balises Html . <SUP></SUB>.
Les quatre instructions Javascript suivantes sont équivalentes :

```
str="Something";
document.write("<SUP>"+str+"</SUP>");
document.write("<SUP>Something</SUP>");
document.write(str.sup());
document.write("Something".sup());
```

+6.6 Les instructions de formatage de document

Rappelons tout d'abord que ce qui suit est optionnel et que vous pouvez utiliser l'instruction document.write() de façon tout à fait classique.
Soit document.write("<BODY BGCOLOR="#FFFFFF");

6.6.1document.bgColor

Cette instruction permet de spécifier la couleur d'arrière-plan d'un objet document. On peut employer le nom ou la valeur RGB de la couleur.

```
document.bgColor="white";
document.bgColor="#FFFFFF";
```

6.6.2 document.fgColor

Cette instruction permet de spécifier la couleur d'avant-plan (texte) d'un objet document. On peut employer le nom ou la valeur RGB de la couleur.

```
document.fgColor="black";
document.fgColor="#000000";
```

6.6.3 document.alinkColor

Cette instruction permet de spécifier la couleur d'un lien actif (après le clic de la souris mais avant de quitter le lien) d'un objet document. On peut employer le nom ou la valeur RGB de la couleur.

```
document.alinkColor="white";
document.alinkColor="#FFFFFF";
```

6.6.4 document.linkColor

Cette instruction permet de spécifier la couleur d'un hyperlien d'un objet document. On peut employer le nom ou la valeur RGB de la couleur.

```
document.linkColor="white";
document.linkColor="#FFFFFF";
```

6.6.5 document.vlinkColor

Cette instruction permet de spécifier la couleur d'un hyperlien déjà visité d'un objet document. On peut employer le nom ou la valeur RGB de la couleur.

```
document.linkColor="white";
document.linkColor="#FFFFFF";
```

Chapitre 7 : Utiliser des variables

7.1 Les variables en Javascript

Les variables contiennent des données qui peuvent être modifiées lors de l'exécution d'un programme. On y fait référence par le nom de cette variable.

Un nom de variable doit commencer par une lettre (alphabet ASCII) ou le signe_ et se composer de lettres, de chiffres et des caractères _ et $ (à l'exclusion du blanc). Le nombre de caractères n'est pas précisé. Pour rappel Javascript est sensible à la case. Attention donc aux majuscules et minuscules!

7.2 La déclaration de variable

Les variables peuvent se déclarer de deux façons :

- soit de façon explicite. On dit à Javascript que ceci est une variable.
  La commande qui permet de déclarer une variable est le mot var. Par exemple :
  var Numero = 1
  var Prenom = "Luc"

- soit de façon implicite. On écrit directement le nom de la variable suivi de la valeur que l'on lui attribue et Javascript s'en accommode. Par exemple :
  Numero = 1
  Prenom = "Luc"

Attention! Malgré cette apparente facilité, la façon dont on déclare la variable aura une grande importance pour la "visibilité" de la variable dans le programme Javascript. Voir à ce sujet, la distinction entre variable locale et variable globale dans le Javascript avancé de ce chapitre.

Pour la clarté de votre script et votre facilité, on ne peut que conseiller d'utiliser à chaque fois le mot var pour déclarer une variable.

7.3 Les données sous Javascript

Javascript utilise 4 types de données :

Type	Description
Des nombres	Tout nombre entier ou avec virgule tel que 22 ou 3.1416
Des chaînes de caractères	Toute suite de caractères comprise entre guillemets telle que "suite de caractères"
Des booléens	Les mots true pour vrai et fase pour faux
Le mot null	Mot spécial qui représente pas de valeur

Notons aussi que contrairement au langage C ou C++, Il ne faut pas déclarer le type de données d'une variable. On n'a donc pas besoin de int, float, double, char et autres long en Javascript.

7.4 Exercice

Nous allons employer la méthode write() pour afficher des variables. On définit une variable appelée texte qui contient une chaîne de caractères "Mon chiffre préféré est " et une autre appelée variable qui est initialisée à 7.

```
<HTML>
<BODY>
<SCRIPT LANGUAGE="Javascript">
<!--
var texte = "Mon chiffre préféré est le "
var variable = 7
document.write(texte + variable);
//-->
</SCRIPT>
</BODY>
</HTML>
```

Le résultat se présente comme suit :
Mon chiffre préféré est le 7

+7.5 Les noms réservés

Les mots de la liste ci-après ne peuvent être utilisés pour des noms de fonctions et de variables
Certains de ces mots sont des mots clés Javascript, d'autres ont été réservés par Netscape pour un futur usage éventuel.

A	abstract
B	boolean break byte
C	case catch char class const continue
D	default do double
E	else extends
F	false final finally float for function
G	goto
I	if implements import in instanceof int interface
L	long
N	native new null
P	package private protected public
R	return
S	short static super switch synchronized
T	this throw throws transient true try
V	var void
W	while with

+7.6 Variables globales et variables locales

Les variables déclarées tout au début du script, en dehors et avant toutes fonctions (voir plus loin...), seront toujours globales, qu'elles soient déclarées avec var ou de façon contextuelle. On pourra donc les exploiter partout dans le script.

Dans une fonction, une variable déclarée par le mot clé var aura une portée limitée à cette seule fonction. On ne pourra donc pas l'exploiter ailleurs dans le script. D'où son nom de locale. Par contre, toujours dans une fonction, si la variable est déclarée contextuellement (sans utiliser le mot var), sa portée sera globale.

Nous reviendrons sur tout ceci dans l'étude des fonctions.

Chapitre 8 : Les opérateurs

Les variables, c'est bien mais encore faut-il pouvoir les manipuler ou les évaluer. Voyons (et ce n'est peut-être pas le chapitre le plus marrant de ce tutorial) les différents opérateurs mis à notre disposition par Javascript.

8.1 Les opérateurs de calcul

Dans les exemples, la valeur initiale de x sera toujours égale à 11

Signe	Nom	Signification	Exemple	Résultat
+	plus	addition	x + 3	14
-	moins	soustraction	x - 3	8
*	multiplié par	multiplication	x*2	22
/	divisé	par division	x /2	5.5
%	modulo	reste de la division par	x%5	1
=	a la valeur	affectation	x=5	5

8.2 Les opérateurs de comparaison

Signe	Nom	Exemple	Résultat
==	égal	x==11	true
<	inférieur	x<11	false
<=	inférieur ou égal	x<=11	true
>	supérieur	x>11	false
=<	supérieur ou égal	x>=11	true
!=	différent	x!=11	false

Important. On confond souvent le = et le == (deux signes =). Le = est un opérateur d'attribution de valeur tandis que le == est un opérateur de comparaison. Cette confusion est une source classique d'erreur de programmation.

8.3 Les opérateurs associatifs

On appelle ainsi les opérateurs qui réalisent un calcul dans lequel une variable intervient des deux côtés du signe = (ce sont donc en quelque sorte également des opérateurs d'attribution).

Dans les exemples suivants x vaut toujours 11 et y aura comme valeur 5.

Signe	Description	Exemple	Signification	Résultat
+=	plus égal	x += y	x = x + y	16
-=	moins égal	x -= y	x = x - y	6
*=	multiplié égal	x *= y	x = x * y	55
/=	divisé égal	x /= y	x = x / y	2.2

8.4 Les opérateurs logiques

Aussi appelés opérateurs booléens, ses opérateurs servent à vérifier deux ou plusieurs conditions.

Signe	Nom	Exemple	Signification
&&	et	(condition1) && (condition2)	condition1 et condition2
||	ou	(condition1) || (condition2)	condition1 ou condition2

8.5 Les opérateurs d'incrémentation

Ces opérateurs vont augmenter ou diminuer la valeur de la variable d'une unité. Ce qui sera fort utile, par exemple, pour mettre en place des boucles.
Dans les exemples x vaut 3.

Signe	Description	Exemple	Signification	Résultat
x++	incrémentation (x++ est le même que x=x+1)	y = x++	3 puis plus 1	4
x--	décrémentation (x-- est le même que x=x-1)	y= x--	3 puis moins 1	2

+8.6 La priorité des opérateurs Javascript

Les opérateurs s'effectuent dans l'ordre suivant de priorité (du degré de priorité le plus faible ou degré de priorité le plus élevé).
Dans le cas d'opérateurs de priorité égale, de gauche à droite.

Opération	Opérateur

,	virgule ou séparateur de liste
= += -= *= /= %=	affectation
? :	opérateur conditionnel
||	ou logique
&&	et logique
== !=	égalité
< <= >= >	relationnel
+ -	addition soustraction
* /	multiplier diviser
! - ++ --	unaire
( )	parenthèses

Chapitre 9 : Les fonctions

9.1 Définition

Une fonction est un groupe de ligne(s) de code de programmation destiné à exécuter une tâche bien spécifique et que l'on pourra, si besoin est, utiliser à plusieurs reprises. De plus, l'usage des fonctions améliorera grandement la lisibilité de votre script.

En Javascript, il existe deux types de fonctions :

- les fonctions propres à Javascript. On les appelle des "méthodes". Elles sont associées à un objet bien particulier comme c'était le cas de la méthode Alert() avec l'objet window.
- les fonctions écrites par vous-même pour les besoins de votre script. C'est à celles-là que nous nous intéressons maintenant.

9.2 Déclaration des fonctions

Pour déclarer ou définir une fonction, on utilise le mot (réservé) function.
La syntaxe d'une déclaration de fonction est la suivante :

```
function nom_de_la_fonction(arguments) {
... code des instructions ...
}
```

Le nom de la fonction suit les mêmes règles que celles qui régissent le nom de variables (nombre de caractères indéfini, commencer par une lettre, peuvent inclure des chiffres...). Pour rappel, Javascript est sensible à la case. Ainsi fonction() ne sera pas égal à Fonction(). En outre, Tous les noms des fonctions dans un script doivent être uniques.

La mention des arguments est facultative mais dans ce cas les parenthèses doivent rester. C'est d'ailleurs grâce à ces parenthèses que l'interpréteur Javascript distingue les variables des fonctions. Nous reviendrons plus en détail sur les arguments et autres paramètres dans la partie Javascript avancé.

Lorsque une accolade est ouverte, elle doit impérativement, sous peine de message d'erreur, être refermée. Prenez la bonne habitude de fermer directement vos accolades et d'écrire votre code entre elles.

Le fait de définir une fonction n'entraîne pas l'exécution des commandes qui la composent. Ce n'est que lors de l'appel de la fonction que le code de programme est exécuté.

9.3 L'appel d'une fonction

L'appel d'une fonction se fait le plus simplement du monde par le nom de la fonction (avec les parenthèses).

Soit par exemple nom_de_la_fonction();

Il faudra veuiller en toute logique (car l'interpréteur lit votre script de haut vers le bas) que votre fonction soit bien définie avant d'être appelée.

9.4 Les fonctions dans <HEAD>...<HEAD>

Il est donc prudent ou judicieux de placer toutes les déclarations de fonction dans l'en-tête de votre page c.-à-d .dans la balise <HEAD> ... <HEAD>. Vous serez ainsi assuré que vos fonctions seront déjà prises en compte par l'interpréteur avant qu'elles soient appelées dans le <BODY>.

9.5 Exemple

Dans cet exemple, on définit dans les balises HEAD, une fonction appelée message() qui affiche le texte "Bienvenue à ma page". cette fonction sera appelée au chargement de la page voir onLoad=.... dans le tag <BODY>.

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="Javascript">
<--
function message() {
document.write("Bienvenue à ma page");
}
//-->
</SCRIPT>
</HEAD>
<BODY onLoad="message()">
</BODY>
</HTML>
```

+9.6 Passer une valeur à une fonction

On peut passer des valeurs ou paramètres aux fonctions Javascript. La valeur ainsi passée sera utilisée par la fonction.
Pour passer un paramètre à une fonction, on fournit un nom d'une variable dans la déclaration de la fonction.

Un exemple un peu simplet pour comprendre. J'écris une fonction qui affiche une boite d'alerte dont le texte peut changer.
Dans la déclaration de la fonction, on écrit :

```
function Exemple(Texte) {
alert(texte);
}
```

Le nom de la variable est Texte et est définie comme un paramètre de la fonction.
Dans l'appel de la fonction, on lui fournit le texte :

```
Exemple("Salut à tous");
```

+9.7 Passer plusieurs valeurs à une fonction

On peut passer plusieurs paramètres à une fonction. Comme c'est souvent le cas en Javascript, on sépare les paramètres par des virgules.

```
function nom_de_la_fonction(arg1, arg2, arg3) {
... code des instructions ...
}
```

Notre premier exemple devient pour la déclaration de fonction :
function Exemplebis(Texte1, Texte2){...}

et pour l'appel de la fonction
Exemplebis("Salut à tous", "Signé Luc")

+9.8 Retourner une valeur

Le principe est simple (la pratique parfois moins). Pour renvoyer un résultat, il suffit d'écrire le mot clé return suivi de l'expression à renvoyer. Notez qu'il ne faut pas entourer l'expression de parenthèses. Par exemple :

```
function cube(nombre) {
var cube = nombre*nombre*nombre
return cube;
}
```

Précisons que l'instruction return est facultative et qu'on peut trouver plusieurs return dans une même fonction.

Pour exploiter cette valeur de la variable retournée par la fonction, on utilise une formulation du type document.write(cube(5)).

+9.9 Variables locales et variables globales

Avec les fonctions, le bon usage des variables locales et globales prend toute son importance.

Une variable déclarée dans une fonction par le mot clé var aura une portée limitée à cette seule fonction. On ne pourra donc pas l'exploiter ailleurs dans le script. On l'appelle donc variable locale.

```
function cube(nombre) {
var cube = nombre*nombre*nombre
}
```

Ainsi la variable cube dans cet exemple est une variable locale. Si vous y faites référence ailleurs dans le script, cette variable sera inconnue pour l'interpréteur Javascript (message d'erreur).

Si la variable est déclarée contextuellement (sans utiliser le mot var), sa portée sera globale -- et pour être tout à fait précis, une fois que la fonction aura été exécutée--.

```
function cube(nombre) {
cube = nombre*nombre*nombre
}
```

La variable cube déclarée contextuellement sera ici une variable globale.

Les variables déclarées tout au début du script, en dehors et avant toutes fonctions, seront toujours globales, qu'elles soient déclarées avec var ou de façon contextuelle.

```
<SCRIPT LANGUAGE="javascript">
var cube=1
function cube(nombre) {
var cube = nombre*nombre*nombre
}
</SCRIPT>
```

La variable cube sera bien globale.

Pour la facilité de gestion des variables, on ne peut que conseiller de les déclarer en début de script (comme dans la plupart des langages de programmation). Cette habitude vous met à l'abri de certaines complications.

Chapitre 10 : Les événements

10.1 Généralités

Avec les événements et surtout leur gestion, nous abordons le côté "*magique*" de Javascript.

En Html classique, il y a un événement que vous connaissez bien. C'est le clic de la souris sur un lien pour vous transporter sur une autre page Web. Hélas, c'est à peu près le seul. Heureusement, Javascript va en ajouter une bonne dizaine, pour votre plus grand plaisir.

Les événements Javascript, associés aux fonctions, aux méthodes et aux formulaires, ouvrent grand la porte pour une réelle *interactivité* de vos pages.

10.2 Les événements

Passons en revue différents événements implémentés en Javascript.

Description	Evénement
Lorsque l'utilisateur clique sur un bouton, un lien ou tout autre élément.	Clik
Lorsque la page est chargée par le browser ou le navigateur.	Load
Lorsque l'utilisateur quitte la page.	Unload
Lorsque l'utilisateur place le pointeur de la souris sur un lien ou tout autre élément.	MouseOver
Lorsque le pointeur de la souris quitte un lien ou tout autre élément.	MouseOut
Attention : Javascript 1.1 (donc pas sous MSIE 3.0 et Netscape 2).	
Lorsque un élément de formulaire a le focus c-à-d devient la zone d'entrée active.	Focus
Lorsque un élément de formulaire perd le focus c-à-d que l'utilisateur clique hors du champs et que la zone d'entrée n'est plus active.	Blur
Lorsque la valeur d'un champ de formulaire est modifiée.	Change
Lorsque l'utilisateur sélectionne un champ dans un élément de formulaire.	Select
Lorsque l'utilisateur clique sur le bouton Submit pour envoyer un formulaire.	Submit

10.3 Les gestionnaires d'événements

Pour être efficace, il faut qu'à ces événements soient associées les actions prévues par vous. C'est le rôle des gestionnaires d'événements. La syntaxe est

onévénement="fonction()"

Par exemple, onClick="alert('Vous avez cliqué sur cet élément')".
De façon littéraire, au clic de l'utilisateur, ouvrir une boite d'alerte avec le message indiqué.

10.3.1 onclick

Evénement classique en informatique, le clic de la souris.

Le code de ceci est :

```
<FORM>
<INPUT TYPE="button" VALUE="Cliquez ici" onClick="alert('Vous avez bien cliqué ici')">
</FORM>
```

Nous reviendrons en détail sur les formulaires dans le chapitre suivant.

10.3.2 onLoad et onUnload

L'événement Load survient lorsque la page a fini de se charger. A l'inverse, Unload survient lorsque l'utilisateur quitte la page.

Les événements onLoad et onUnload sont utilisés sous forme d'attributs de la balise <BODY> ou <FRAMESET>. On peut ainsi écrire un script pour souhaiter la bienvenue à l'ouverture d'une page et un petit mot d'au revoir au moment de quitter celle-ci.

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE='Javascript'>
function bienvenue() {
alert("Bienvenue à cette page");
}
function au_revoir() {
alert("Au revoir");
}
</SCRIPT>
</HEAD>
<BODY onLoad='bienvenue()' onUnload='au_revoir()'>
Html normal
</BODY>
</HTML>
```

10.3.4 onmouseOver et onmouseOut

L'événement onmouseOver se produit lorsque le pointeur de la souris passe au dessus (sans cliquer) d'un lien ou d'une image. Cet événement est fort pratique pour, par exemple, afficher des explications soit dans la barre de statut soit avec une petite fenêtre genre infobulle.

L'événement onmouseOut, généralement associé à un onmouseOver, se produit lorsque le pointeur quitte la zone sensible (lien ou image).
Notons qui si onmouseOver est du Javascript 1.0, onmouseOut est du Javascript 1.1.
En clair, onmouseOut ne fonctionne pas avec Netscape 2.0 et Explorer 3.0.

10.3.5 onFocus

L'événement onFocus survient lorsqu'un champ de saisie a le focus c.-à-d. quand son emplacement est prêt à recevoir ce que l'utilisateur à l'intention de taper au clavier. C'est souvent la conséquence d'un clic de souris ou de l'usage de la touche "Tab".

10.3.6 onBlur

L'événement onBlur a lieu lorsqu'un champ de formulaire perd le focus. Cela se produit quand l'utilisateur ayant terminé la saisie qu'il effectuait dans une case, clique en dehors du champ ou utilise la touche "Tab" pour passer à un champ. Cet événement sera souvent utilisé pour vérifier la saisie d'un formulaire.

Le code est :

```
<FORM>
<INPUT TYPE=text onBlur="alert('Ceci est un Blur')">
</FORM>
```

10.3.7 onchange

Cet événement s'apparente à l'événement onBlur mais avec une petite différence. Non seulement la case du formulaire doit avoir perdu le focus mais aussi son contenu doit avoir été modifié par l'utilisateur.

10.3.8 onselect

Cet événement se produit lorsque l'utilisateur a sélectionné (mis en surbrillance ou en vidéo inverse) tout ou partie d'une zone de texte dans une zone de type text ou textarea.

+10.4 Gestionnaires d'événement disponibles en Javascript

Il nous semble utile dans cette partie "avancée" de présenter la liste des objets auxquels correspondent des gestionnaires d'événement bien déterminés.

Objets	Gestionnaires d'événement disponibles
Fenêtre	onLoad, onUnload
Lien hypertexte	onClick, onmouseOver, on mouseOut
Elément de texte	onBlur, onChange, onFocus, onSelect
Elément de zone de texte	onBlur, onChange, onFocus, onSelect
Elément bouton	onClick
Case à cocher	onClick
Bouton Radio	onClick
Liste de sélectionon	Blur, onChange, onFocus
Bouton Submit	onClick
Bouton Reset	onClick

+10.5 La syntaxe de onmouseOver

Le code du gestionnaire d'événement onmouseOver s'ajoute aux balises de lien :

<A HREF="" onmouseOver="action()">lien</A>

Ainsi, lorsque l'utilisateur passe avec sa souris sur le lien, la fonction action() est appelée. L'attribut HREF est indispensable. Il peut contenir l'adresse d'une page Web si vous souhaitez que le lien soit actif ou simplement des guillemets si aucun lien actif n'est prévu. Nous reviendrons ci-après sur certains désagréments du codage HREF="".

Voici un exemple. Par le survol du lien "message important", une fenêtre d'alerte s'ouvre.

Le code est :

```
<BODY>
...
<A HREF="" onmouseOver="alert('Coucou')">message important</A>
...
<BODY>
```

ou si vous préférez utiliser les balises <HEAD>

```
<HTML>
<HEAD>
<SCRIPT language="Javascript">
function message(){
alert("Coucou")
}
</SCRIPT>
</HEAD>
<BODY>
<A HREF="" onmouseOver="message()">message important</A>
</BODY>
</HTML>
```

+10.6 La syntaxe de onmouseOut

Tout à fait similaire à onmouseOver, sauf que l'événement se produit lorsque le pointeur de la souris quitte le lien ou la zone sensible.

Au risque de nous répéter, si onmouseOver est du Javascript 1.0 et sera donc reconnu par tous les browsers, onmouseOut est du Javascript 1.1 et ne sera reconnu que par Netscape 3.0 et plus et Explorer 4.0 et plus (et pas par Netscape 2.0 et Explorer 3.0)

On peut imaginer le code suivant :
<A HREF="" onmouseOver="alert('Coucou')" onmouseOut="alert('Au revoir')">message important</A>

Les puristes devront donc prévoir une version différente selon les versions Javascript.

+10.7 Problème! Et si on clique quand même...

Vous avez codé votre instruction onmouseOver avec le lien fictif <A HREF=""... >, vous avez même prévu un petit texte, demandant gentiment à l'utilisateur de ne pas cliquer sur le lien et comme de bien entendu celui-ci clique quand même.
Horreur, le browser affiche alors l'entièreté des répertoires de sa machine ou de votre site). Ce qui est un résultat non désiré et pour le moins imprévu.

Pour éviter cela, prenez l'habitude de mettre l'adresse de la page encours ou plus simplement le signe # (pour un ancrage) entre les guillemets de HREF. Ainsi, si le lecteur clique quand même sur le lien, au pire, la page encours sera simplement rechargée et sans perte de temps car elle est déjà dans le cache du navigateur.
Prenez donc l'habitude de mettre le code suivant <A HREF="#" onmouseOver="action()"> lien </A>.

+10.8 Changement d'images

Avec le gestionnaire d'événement onmouseOver, on peut prévoir qu'après le survol d'un image par l'utilisateur, une autre image apparaisse (pour autant qu'elle soit de la même taille).
le code est relativement simple.

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="Javascript1.1">
function lightUp() {
document.images["homeButton"].src="button_hot.gif"
}
function dimDown() {
document.images["homeButton"].src="button_dim.gif"
}
</SCRIPT>
</HEAD>
<BODY>
<A HREF="#" onmouseOver="lightUp();" onmouseOut="dimDown();">
<IMG SRC="button_dim.gif" name="homeButton" width=100 height=50 border=0> </A>
</BODY>
</HTML>
```

Compléter toujours en Javascript les attributs width=x height=y de vos images.

Il n'y a pas d'exemple ici pour la compatibilité avec les lecteurs utilisant explorer 3.0 en effet, non seulement onmouseOut mais aussi image[] est du Javascript 1.1.

+10.9 L'image invisible

Ce changement d'image ne vous donne-t-il pas des idées?... Petit futé! Et oui, on peut prévoir une image invisible de la même couleur que l'arrière plan (même transparente). On la place avec malice sur le chemin de la souris de l'utilisateur et son survol peut ,à l'insu de l'utilisateur, déclencher un feu d'artifice d'actions de votre choix. Magique le Javascript ?

Chapitre 11 : Les conditions

11.1 Si Maman si ..." ou l'expression if

A un moment ou à un autre de la programmation, on aura besoin de tester une condition. Ce qui permettra d'exécuter ou non une série d'instructions.

Dans sa formulation la plus simple, l'expression if se présente comme suit
if (condition vraie) {
une ou plusieurs instructions;
}

Ainsi, si la condition est vérifiée, les instructions s'exécutent. Si elle ne l'est pas, les instructions ne s'exécutent pas et le programme passe à la commande suivant l'accolade de fermeture.

De façon un peu plus évoluée, il y a l'expression if...else

```
if (condition vraie) {
instructions1;
}
else {
instructions2;
}
```

Si la condition est vérifiée (true), le bloc d'instructions 1 s'exécute. Si elle ne l'est pas (false), le bloc d'instructions 2 s'exécute.

Dans le cas où il n'y a qu'une instruction, les accolades sont facultatives.

Grâce aux opérateurs logiques "et" et "ou", l'expression de test pourra tester une association de conditions. Ainsi if ((condition1) && (condition2)), testera si la condition 1 et la condition 2 est réalisée. Et if ((condition1) || (condition2)), testera si une au moins des conditions est vérifiée.

Pour être complet (et pour ceux qui aiment les écritures concises), il y a aussi :

(expression) ? instruction a : instruction b

Si l'expression entre parenthèse est vraie, l'instruction a est exécutée.Si l'expression entre parenthèses retourne faux, c'est l'instruction b qui est exécutée.

11.2 L'expression for

L'expression for permet d'exécuter un bloc d'instructions un certain nombre de fois en fonction de la réalisation d'un certain critère. Sa syntaxe est :

```
for (valeur initiale ; condition ; progression) {
instructions;
}
```

Prenons un exemple concret

```
for (i=0, i<10, i++) {
document.write(i + "<BR>")
}
```

Au premier passage, la variable i, étant initialisée à 0, vaut bien entendu 0. Elle est bien inférieure à 10. Elle est donc incrémentée d'une unité par l'opérateur d'incrémentation i++ (i vaut alors 2) et les instructions s'exécutent.
A la fin de l'exécution des instructions, on revient au compteur. La variable i (qui vaut 2) est encore toujours inférieure à 10. Elle est augmentée de 1 et les instructions sont à nouveau exécutées. Ainsi de suite jusqu'à ce que i vaille 10. La variable i ne remplit plus la condition i<10. La boucle s'interrompt et le programme continue après l'accolade de fermeture.

+11.3 While

L'instruction while permet d'exécuter une instruction (ou un groupe d'instructions) un certain nombre de fois.

```
while (condition vraie){
continuer à faire quelque chose
}
```

Aussi longtemps que la condition est vérifiée, Javascript continue à exécuter les instructions entre les accolades. Une fois que la condition n'est plus vérifiée, la boucle est interrompue et on continue le script.

Prenons un exemple.

```
compt=1;
while (compt<5) {
document.write ("ligne : " + compt + "<BR>");
compt++;
}
document.write("fin de la boucle");
```

Voyons comment fonctionne cet exemple. D'abord la variable qui nous servira de compteur compt est initialisée à 1. La boucle while démarre donc avec la valeur 1 qui est inférieure à 5. La condition est vérifiée. On exécute les instructions des accolades. D'abord, "ligne : 1" est affichée et ensuite le compteur est incrémenté de 1 et prend donc la valeur 2. La condition est encore vérifiée. Les instructions entre les accolades sont exécutées. Et ce jusqu'à l'affichage de la ligne 4. Là, le compteur après l'incrémentation vaut 5. La condition n'étant plus vérifiée, on continue dans le script et c'est alors fin de boucle qui est affiché.

Attention ! Avec ce système de boucle, le risque existe (si la condition est toujours vérifiée), de faire boucler indéfiniment l'instruction. Ce qui à la longue fait misérablement planter le browser !

Ce qui donnerait à l'écran :

```
ligne : 1
ligne : 2
ligne : 3
ligne : 4
fin de la boucle
```

+11.4 Break

L'instruction break permet d'interrompre prématurément une boucle for ou while.

Pour illustrer ceci, reprenons notre exemple :

```
compt=1;
while (compt<5) {
if (compt == 4)
break;
document.write ("ligne : " + compt + "<BR>");
```

```
compt++;
}
document.write("fin de la boucle");
```

Le fonctionnement est semblable à l'exemple précédent sauf lorsque le compteur vaut 4. A ce moment, par le break, on sort de la boucle et "fin de boucle" est affiché.

Ce qui donnerait à l'écran :

```
ligne : 1
ligne : 2
ligne : 3
fin de la boucle
```

+11.5 Continue

L'instruction continue permet de sauter une instruction dans une boucle for ou while et de continuer ensuite le bouclage (sans sortir de celui-ci comme le fait break).

Reprenons notre exemple ;

```
compt=1;
while (compt<5) {
if (compt == 3){
compt++
continue;}
document.write ("ligne : " + compt + "<BR>");
compt++;
}
document.write("fin de la boucle");
```

Ici, la boucle démarre. Lorsque le compteur vaut 3, par l'instruction continue, on saute l'instruction document.write (ligne : 3 n'est pas affichée) et on continue la boucle. Notons qu'on a dû ajouter compt++ avant continue; pour éviter un bouclage infini et un plantage du navigateur (compt restant à 3).

Ce qui fait à l'écran :

```
ligne : 1
ligne : 2
ligne : 4
fin de la boucle
```

Chapitre 12 : Les formulaires

12.1 Généralités

Avec Javascript, les formulaires Html prennent une toute autre dimension. N'oublions pas qu'en Javascript, on peut accéder à chaque élément d'un formulaire pour, par exemple, y aller lire ou écrire une valeur, noter un choix auquel on pourra associer un gestionnaire d'événement... Tous ces éléments renforceront grandement les capacités interactives de vos pages.

Mettons au point le vocabulaire que nous utiliserons. Un formulaire est l'élément Html déclaré par les balises <FORM></FORM>. Un formulaire contient un ou plusieurs éléments que nous appellerons des contrôles (widgets). Ces contrôles sont notés par exemple par la balise <INPUT TYPE= ...>.

12.2 Déclaration d'un formulaire

La déclaration d'un formulaire se fait par les balises <FORM> et </FORM>. Il faut noter qu'en Javascript, l'attribut NAME="nom_du_formulaire" a toute son importance pour désigner le chemin complet des éléments. En outre, les attributs ACTION et METHOD sont facultatifs pour autant que vous ne faites pas appel au serveur.

Une erreur classique en Javascript est, emporté par son élan, d'oublier de déclarer la fin du formulaire </FORM> après avoir incorporé un contrôle.

12.3 Le contrôle ligne de texte

La zone de texte est l'élément d'entrée/sortie par excellence de Javascript. La syntaxe Html est <INPUT TYPE="text" NAME="nom" SIZE=x MAXLENGTH=y> pour un champ de saisie d'une seule ligne, de longueur x et de longueur maximale de y.

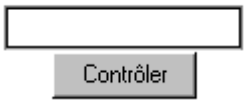
L'objet text possède trois propriétés :

Propriété	Description
name	indique le nom du contrôle par lequel on pourra accéder.
Defaultvalue	indique la valeur par défaut qui sera affichée dans la zone de texte.
Value	indique la valeur en cours de la zone de texte. Soit celle tapée par l'utilisateur ou si celui-ci n'a rien tapé, la valeur par défaut.

12.3.1 Lire une valeur dans une zone de texte

Voici un exemple que nous détaillerons :

Voici une zone de texte. Entrez une valeur et appuyer sur le bouton pour contrôler celle-ci.



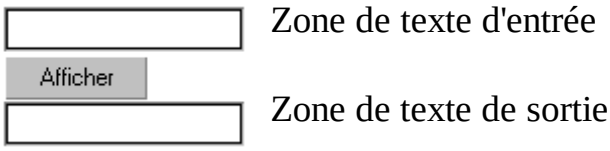
Le script complet est celui-ci :

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="javascript">
function controle(form1) {
var test = document.form1.input.value;
alert("Vous avez tapé : " + test);
}
</SCRIPT>
</HEAD>
<BODY>
<FORM NAME="form1">
<INPUT TYPE="text" NAME="input" VALUE=""><BR>
<INPUT TYPE="button" NAME="bouton" VALUE="Contrôler" onClick="controle(form1)">
</FORM>
</BODY>
</HTML>
```

Lorsqu'on clique le bouton "contrôler", Javascript appelle la fonction controle() à laquelle on passe le formulaire dont le nom est form1 comme argument.
Cette fonction controle() définie dans les balises <HEAD> prend sous la variable test, la valeur de la zone de texte. Pour accéder à cette valeur, on note le chemin complet de celle-ci (voir le chapitre "Un peu de théorie objet"). Soit dans le document présent, il y a l'objet formulaire appelé form1 qui contient le contrôle de texte nommé input et qui a comme propriété l'élément de valeur value. Ce qui donne document.form1.input.value.

12.3.2 Ecrire une valeur dans une zone de texte

Entrez une valeur quelconque dans la zone de texte d'entrée. Appuyer sur le bouton pour afficher cette valeur dans la zone de texte de sortie.



Voici le code :

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="javascript">
function afficher(form2) {
var testin =document. form2.input.value;
document.form2.output.value=testin
}
</SCRIPT>
</HEAD>
<BODY>
<FORM NAME="form2">
<INPUT TYPE="text" NAME="input" VALUE=""> Zone de texte d'entrée <BR>
<INPUT TYPE="button" NAME="bouton" VALUE="Afficher" onClick="afficher(form2)"><BR>
<INPUT TYPE="text" NAME="output" VALUE=""> Zone de texte de sortie
</FORM>
</BODY>
</HTML>
```

Lorsqu'on clique le bouton "Afficher", Javascript appelle la fonction afficher() à laquelle on passe le formulaire dont le nom est cette fois form2 comme argument.
Cette fonction afficher() définie dans les balises <HEAD> prend sous la variable testin, la valeur de la zone de texte d'entrée. A l'instruction suivante, on dit à Javascript que la valeur de la zone de texte output comprise dans le formulaire nommé form2 est celle de la variable testin. A nouveau, on a utilisé le chemin complet pour arriver à la propriété valeur de l'objet souhaité soit en Javascript document.form2.output.value.

12.4 Les boutons radio

Les boutons radio sont utilisés pour noter un choix, et seulement un seul, parmi un ensemble de propositions.

Propriété	Description
name	indique le nom du contrôle. Tous les boutons portent le même nom.
index	l'index ou le rang du bouton radio en commençant par 0.
checked	indique l'état en cours de l'élément radio
defaultchecked	indique l'état du bouton sélectionné par défaut.
value	indique la valeur de l'élément radio.

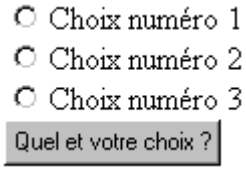
Prenons un exemple :

```
<HTML>
<HEAD>
<SCRIPT language="javascript">
function choixprop(form3) {
if (form3.choix[0].checked) { alert("Vous avez choisi la proposition " + form3.choix[0].value) };
if (form3.choix[1].checked) { alert("Vous avez choisi la proposition " + form3.choix[1].value) };
if (form3.choix[2].checked) { alert("Vous avez choisi la proposition " + form3.choix[2].value) };
}
```

```
</SCRIPT>
</HEAD>
<BODY>
Entrez votre choix :
<FORM NAME="form3">
<INPUT TYPE="radio" NAME="choix" VALUE="1">Choix numéro 1<BR>
<INPUT TYPE="radio" NAME="choix" VALUE="2">Choix numéro 2<BR>
<INPUT TYPE="radio" NAME="choix" VALUE="3">Choix numéro 3<BR>
<INPUT TYPE="button"NAME="but" VALUE="Quel et votre choix ?" onClick="choixprop(form3)">
</FORM>
</BODY>
</HTML>
```

PS: Ce programme a été écrit avec un souci didactique. On pourrait l'écrire avec des codes plus compacts.

Entrez votre choix :



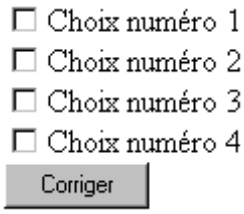
Dans le formulaire nommé form3, on déclare trois boutons radio. Notez que l'on utilise le même nom pour les trois boutons. Vient ensuite un bouton qui déclenche la fonction choixprop().
Cette fonction teste quel bouton radio est coché. On accède aux boutons sous forme d'indice par rapport au nom des boutons radio soit choix[0], choix[1], choix[2]. On teste la propriété checked du bouton par if(form3.choix[x].checked). Dans l'affirmative, une boite d'alerte s'affiche. Ce message reprend la valeur attachée à chaque bouton par le chemin form3.choix[x].value.

12.5 Les boutons case à cocher (checkbox)

Les boutons case à cocher sont utilisés pour noter un ou plusieurs choix (pour rappel avec les boutons radio un seul choix) parmi un ensemble de propositions. A part cela, sa syntaxe et son usage est tout à fait semblable aux boutons radio sauf en ce qui concerne l'attribut name.

Propriété	Description
name	indique le nom du contrôle. Toutes les cases à cocher portent un nom différent.
checked	indique l'état en cours de l'élément case à cocher.
defaultchecked	indique l'état du bouton sélectionné par défaut.
value	indique la valeur de l'élément case à cocher.

Entrez votre choix :
Il faut sélectionner les numéros 1,2 et 4 pour avoir la bonne réponse.




```
<HTML>
<HEAD>
<script language="javascript">
function reponse(form4) {
```

```
if ( (form4.check1.checked) == true && (form4.check2.checked) == true && (form4.check3.checked) == false
&& (form4.check4.checked) == true)
{ alert("C'est la bonne réponse! ") }
else
{alert("Désolé, continuez à chercher.")}
}
</SCRIPT>
</HEAD>
<BODY>
Entrez votre choix :
<FORM NAME="form4">
<INPUT TYPE="CHECKBOX" NAME="check1" VALUE="1">Choix numéro 1<BR>
<INPUT TYPE="CHECKBOX" NAME="check2" VALUE="2">Choix numéro 2<BR>
<INPUT TYPE="CHECKBOX" NAME="check3" VALUE="3">Choix numéro 3<BR>
<INPUT TYPE="CHECKBOX" NAME="check4" VALUE="4">Choix numéro 4<BR>
<INPUT TYPE="button"NAME="but" VALUE="Corriger" onClick="reponse(form4)">
</FORM>
</BODY>
</HTML>
```

Dans le formulaire nommé form4, on déclare quatre cases à cocher. Notez que l'on utilise un nom différent pour les quatre boutons. Vient ensuite un bouton qui déclenche la fonction reponse().
Cette fonction teste quelles cases à cocher sont sélectionnées. Pour avoir la bonne réponse, il faut que les cases 1, 2 et 4 soient cochées. On accède aux cases en utilisant chaque fois leur nom. On teste la propriété checked du bouton par (form4.nom_de_la_case.checked). Dans l'affirmative (&& pour et logique), une boite d'alerte s'affiche pour la bonne réponse. Dans la négative, une autre boite d'alerte vous invite à recommencer.

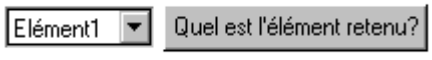
12.6 Liste de sélection

Le contrôle liste de sélection vous permet de proposer diverses options sous la forme d'une liste déroulante dans laquelle l'utilisateur peut cliquer pour faire son choix. Ce choix reste alors affiché.
La boite de la liste est crée par la balise <SELECT> et les éléments de la liste par un ou plusieurs tags <OPTION>. La balise </SELECT> termine la liste.

Propriété	Description
name	indique le nom de la liste déroulante.
length	indique le nombre d'éléments de la liste. S'il est indiqué dans le tag <SELECT>, tous les éléments de la liste seront affichés. Si vous ne l'indiquez pas un seul apparaîtra dans la boite de la liste déroulante.
selectedIndex	indique le rang à partir de 0 de l'élément de la liste qui a été sélectionné par l'utilisateur.
defaultselected	indique l'élément de la liste sélectionné par défaut. C'est lui qui apparaît alors dans la petite boite.

Un petit exemple comme d'habitude :

Entrez votre choix :




```
<HTML>
<HEAD>
<script language="javascript"> function liste(form5) {
alert("L\'élément " + (form5.list.selectedIndex + 1)); }
</SCRIPT>
</HEAD>
<BODY>
Entrez votre choix : <FORM NAME="form5">
<SELECT NAME="list">
```

<OPTION VALUE="1">Elément 1
<OPTION VALUE="2">Elément 2
<OPTION VALUE="3">Elément 3
</SELECT>
<INPUT TYPE="button"NAME="b" VALUE="Quel est l'élément retenu?" onClick="liste(form5)"> </FORM>
</BODY>
</HTML>

Dans le formulaire nommé form5, on déclare une liste de sélection par la balise <SELECT></SELECT>. Entre ses deux balises, on déclare les différents éléments de la liste par autant de tags <OPTION>. Vient ensuite un bouton qui déclenche la fonction liste().
Cette fonction teste quelle option a été sélectionnée. Le chemin complet de l'élément sélectionné est form5.nomdelaliste.selectedIndex. Comme l'index commence à 0, il ne faut pas oublier d'ajouter 1 pour retrouver le juste rang de l'élément.

+12.7 Le contrôle textarea (pour les bavards)

L'objet textarea est une zone de texte de plusieurs lignes.

La syntaxe Html est :
<FORM>
<TEXTAREA NAME="nom" ROWS=x COLS=y>
texte par défaut
</TEXTAREA>
</FORM>

où ROWS=x représente le nombre de lignes et COLS=y représente le nombre de colonnes.

L'objet textarea possède plusieurs propriétés :

Propriété	Description
name	indique le nom du contrôle par lequel on pourra accéder.
defaultvalue	indique la valeur par défaut qui sera affichée dans la zone de texte.
value	indique la valeur en cours de la zone de texte. Soit celle tapée par l'utilisateur ou si celui-ci n'a rien tapé, la valeur par défaut.

A ces propriétés, il faut ajouter onFocus(), onBlur(), onSelect() et onChange().

En Javascript, on utilisera \r\n pour passer à la ligne.
Comme par exemple dans l'expression document.Form.Text.value = 'Check\r\nthis\r\nout'.

+12.8 Le contrôle Reset

Le contrôle Reset permet d'annuler les modifications apportées aux contrôles d'un formulaire et de restaurer les valeurs par défaut.

la syntaxe Html est :

<INPUT TYPE="reset" NAME="nom" VALUE "texte">
où VALUE donne le texte du bouton.

Une seule méthode est associée au contrôle Reset, c'est la méthode onClick(). Ce qui peut servir, par exemple, pour faire afficher une autre valeur que celle par défaut.

+12.9 Le contrôle Submit

Le contrôle a la tâche spécifique de transmettre toutes les informations contenues dans le formulaire à l'URL désignée dans l'attribut ACTION du tag <FORM>.

la syntaxe Html est :

<INPUT TYPE="submit" NAME="nom" VALUE "texte">
où VALUE donne le texte du bouton.

Une seule méthode est associée au contrôle Submit, c'est la méthode onClick().

+12.10 Le contrôle Hidden (caché)

Le contrôle Hidden permet d'entrer dans le script des éléments (généralement des données) qui n'apparaîtront pas à l'écran. Ces éléments sont donc cachés. D'où son nom.

la syntaxe Html est :

<INPUT TYPE="hidden" NAME="nom" VALUE "les données cachées">

+12.11 L'envoi d'un formulaire par Email.

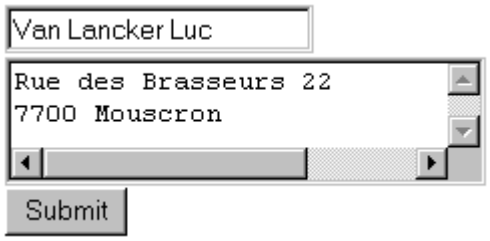
Uniquement Netscape !!!

A force de jouer avec des formulaires, il peut vous prendre l'envie de garder cette source d'information. Mais comment faire? Javascript, et à fortiori le Html, ne permet pas d'écrire dans un fichier . Ensuite, le contrôle Submit est surtout destiné à des CGI ce qui entraîne (encore) un codage spécial à maîtriser. D'autant que pour nous simples et présumés incompétents internautes, la plupart des providers ne permettra pas d'héberger une CGI faite par un amateur pour des raisons (tout à fait compréhensibles) de sécurité. Il ne reste plus que l'unique solution de l'envoi d'un formulaire via le courrier électronique.

La syntaxe est :

<FORM METHOD="post" ACTION="mailto:votre_adresse_Email">
<INPUT TYPE=text NAME="nom">
<TEXTAREA NAME="adresse" ROWS=2 COLS=35>
</TEXTAREA>
<INPUT TYPE=submit VALUE="Submit">
</FORM>

Ce qui donne :



Vous recevrez dans notre boite de réception, un truc bizarre du genre :
nom=Van+Lancker+Luc&adresse=Rue+des+Brasseurs+2217OD%OA7700+Mouscron.
où on retrouve les champs nom= et adresse=, où les champs sont séparés par le signe &, où les espaces sont remplacés par le signe + et 17%OD%OA correspond à un passage à la ligne.

Attention ! Ceci ne marche que sous Netscape et pas sous Microsoft Explorer 3.0 ...
Avec Explorer, le mailto ouvre le programme de Mail mais n'envoie rien du tout.

Chapitre 13 : Un peu de tout

13.1 Les boites de dialogue ou de message

Javascript met à votre disposition 3 boites de message :

- alert()
- prompt()
- confirm()

Ce sont toutes trois des méthodes de l'objet window.

13.2 La méthode alert()

La méthode alert() doit, à ce stade de votre étude, vous être familière car nous l'avons déjà souvent utilisée.

La méthode alert() affiche une boite de dialogue dans laquelle est reproduite la valeur (variable et/ou chaîne de caractères) de l'argument qui lui a été fourni. Cette boite bloque le programme en cours tant que l'utilisateur n'aura pas cliqué sur "OK".
Alert() sera aussi très utile pour vous aider à débbuger les scripts.

Sa syntaxe est :
alert(variable);
alert("chaîne de caractères");
alert(variable + "chaîne de caractères");

Si vous souhaitez écrire sur plusieurs lignes, il faudra utiliser le signe \n.

13.3 La méthode prompt()

Dans le même style que la méthode alert(), Javascript vous propose une autre boite de dialogue, dans le cas présent appelée boite d'invite, qui est composée d'un champ comportant une entrée à compléter par l'utilisateur. Cette entrée possède aussi une valeur par défaut.

La syntaxe est :
prompt("texte de la boite d'invite","valeur par défaut");

En cliquant sur OK, la méthode renvoie la valeur tapée par l'utilisateur ou la réponse proposée par défaut. Si l'utilisateur clique sur Annuler ou Cancel, la valeur null est alors renvoyée.
Prompt() est parfois utilisé pour saisir des données fournies par l'utilisateur. Selon certaines sources, le texte ne doit cependant pas dépasser 45 caractères sous Netscape et 38 sous Explorer 3.0.

13.4 La méthode confirm()

Cette méthode affiche 2 boutons "OK" et "Annuler". En cliquant sur OK, continue() renvoie la valeur true et bien entendu false si on a cliqué sur Annuler. Ce qui peut permettre, par exemple, de choisir une option dans un programme.

La syntaxe de l'exemple est :
confirm("Voulez-vous continuer ?")

+13.5 Une minuterie

Javascript met à votre disposition une minuterie (ou plus précisément un compteur à rebours) qui permettra de déclencher une fonction après un laps de temps déterminé.

La syntaxe de mise en route du temporisateur est :

nom_du_compteur = setTimeout("fonction_appelée()", temps en milliseconde)

Ainsi, setTimeout("demarrer()",5000) va lancer la fonction demarer() après 5 secondes.

Pour arrêter le temporisateur avant l'expiration du délai fixé, il y a :
clearTimeout(nom_du_compteur)

+13.6 L'emploi de this

Pour désigner l'objet en cours, Javascript met à votre disposition le mot-clé this. Cette écriture raccourcie est souvent utilisée (sans risque de confusion) en remplacement du chemin complet de l'objet dans un formulaire. Un exemple vous éclairera mieux qu'un long discours.

Soit un script avec un formulaire :

```
<FORM NAME="form3">
<INPUT TYPE="radio" NAME="choix" VALUE="1">Choix numéro 1<BR>
<INPUT TYPE="radio" NAME="choix" VALUE="2">Choix numéro 2<BR>
<INPUT TYPE="radio" NAME="choix" VALUE="3">Choix numéro 3<BR>
<INPUT TYPE="button"NAME="but" VALUE="Quel et votre choix ?" onClick="choixprop(form3)">
</FORM>
```

Au lieu d'employer choixprop(form3), on aurait pu utiliser choixprop(this.form) et éviter ainsi toute confusion avec les autres noms de formulaires. Dans cet exemple, this.form désigne le formulaire form3 complet. Par contre, choixprop(this) n'aurait désigné que l'élément de type bouton du formulaire form3.

Pour être complet, this est utilisé aussi pour créer une ou plusieurs propriétés d'un objet. Ainsi, pour créer un objet livre avec les propriétés auteur, éditeur et prix cette opération peut être effectuée à l'aide de la fonction :

```
function livre(auteur, editeur, prix) {
this.auteur = auteur;
this.editeur = editeur;
this. prix = prix;
}
```

Chapitre 14 : Les messages d'erreur

14.1 Bon courage !

On ne peut pas dire que les outils de débuggage offerts par Javascript soient des plus évolués. Pour un peu, on se croirait revenu aux temps préhistoriques du Basic. Ainsi, corriger vos erreurs en Javascript ressemble curieusement à une enquête policière digne de Sherlock Holmes ou au jeu des sept erreurs.

En effet, on ne peut se fier à 100% aux précisions données par l'interpréteur. Le fameux "Javascript Error line x" est plutôt à comprendre comme "ça commence à foirer à la ligne x" et il sera parfois nécessaire de remonter de plusieurs lignes dans le script pour trouver l'erreur initiale.

14.2 Les types d'erreurs.

Il y a 3 grandes catégories d'erreurs dans l'utilisation d'un programme Javascript :

- les erreurs au chargement.
- les erreurs à l'exécution.
- les erreurs de logique.

14.3 Les erreurs au chargement

Au chargement du script par le browser, Javascript passe en revue les différentes erreurs qui peuvent empêcher le bon déroulement de celui-ci.
Les erreurs au chargement, nombreuses lors de l'apprentissage de Javascript, sont souvent dues à des fautes de frappe et/ou des erreurs de syntaxe.
Pour vous aider à déterminer l'erreur, Javascript affiche sa fameuse boite de message d'erreur, vous indique le problème et le texte de l'erreur. Ne perdez pas de vue que Javascript ne vous indique pas toujours l'erreur véritable et que selon l'erreur, celle-ci peut se situer bien plus avant dans le script. Des exemples classiques d'erreurs au chargement sont des parenthèses ou des accolades non fermées, des guillemets manquants, etc.

14.4 Les erreurs à l'exécution

Ici votre script se charge sans problème, mais cette satanée boite de message d'erreurs apparaît lorsque l'exécution du script est demandée. Alors que les erreurs au chargement étaient surtout dues au mauvais usage de la syntaxe, les erreurs à l'exécution proviennent d'un mauvais usage des commandes ou des objets Javascript. Un exemple d'erreur à l'exécution est un appel erroné à un une variable ou une fonction inexistante (car il y a, par exemple, une erreur dans le nom de la variable ou de la fonction).

14.5 Les erreurs de logique

Ce sont les plus vicieuses car le "débuggeur" de Javascript ne signale bien entendu aucune erreur et votre script se déroule correctement. Hélas, à l'arrivée, le résultat ne correspond pas à celui espéré.
Il n'y a plus qu'à revoir la construction logique de votre script. Cent fois sur le métier remettez votre ouvrage...

De nombreuses erreurs de logique sont dues à des valeurs de variables incorrectes.
Voici quelques conseils :

- Dans le cas où l'utilisateur doit entrer une valeur, celle-ci était-t-elle au bon format?
- N'est-il pas utile de prévoir un petit script pour vérifier le format d'entrée ?
- On peut ajouter des points de contrôle de valeur de variable ou de passage avec l'instruction alert(variable) ou alerte("Point de passage1").

14.6 Les grands classiques des erreurs.

On peut dresser une liste d'erreurs que tous les débutants (et même certains plus confirmés) font ou feront tôt ou tard.

- Soyez vigilant au nom des variables (case sensitive). Mavariable et mavariable sont deux variables distinctes. Eviter d'utiliser des noms de variables trop rapprochants.
- Le nom de la fonction a-t-il bien la même orthographe dans la déclaration et dans l'appel. Le nom des fonctions est-il bien unique dans le script?
- N'oubliez pas les guillemets ou apostrophes avant et après les chaînes de caractères. •Avez-vous bien mis des virgules entre vos différents paramètres ou arguments?
- Avez-vous placé vos accolades au bon endroit sans avoir oublié de les fermer (surtout dans le cas de blocs de commandes imbriquées).
- Assurez-vous que les noms des objets Javascript sont corrects. Le piège est que les objets Javascript commencent par une majuscule (Date, Math, Array...) mais que les propriétés commencent par une minuscule (alert).
- La confusion entre = opérateur d'affectation et == opérateur de comparaison

14.7 Conclusion

Débugger du Javacript n'est pas toujours la joie mais avec l'habitude de ce genre d'exercice, on s'en sort assez bien. Conscient de ce réel problème, on semble mettre au point chez Netscape et Microsoft des débuggeurs mais nous n'avons pas encore eu l'occasion de les tester.

Chapitre 15 : Mini FAQ

15.1 Pas de changement après modification du script

Il vous arrivera sûrement qu'après une modification (correction?) de votre script, aucun changement ne soit visible à l'écran après avoir fait "Reload" ou "Actualiser".

- Vérifier bien si vous avez enregistré vos modifications (cela arrive même aux meilleurs).
- Il faut parfois recharger plus profondément votre page en faisant Shift + Reload (sous Netscape) ou cliquez dans la zone de localisation du browser et faire Enter.

15.2 Le cas Microsoft Internet Explorer 3.0

Microsoft Explorer 3.0 supporte une forme de Javascript. Microsoft a implanté le langage Javascript séparément et indépendamment de Netscape. Internet Explorer "comprend" le code Javascript, mais son implémentation n'est pas complète. Il faut garder cela à l'esprit lorsque on veut écrire du code Javascript qui doit fonctionner à la fois sous Netscape et Explorer 3.0.

Votre code Javascript sous Explorer doit

- impérativement être du Javascript 1.0. Toutes les fonctions Javascript 1.1, si familières pour Netscape 3, entraîneront des messages d'erreur.
- ne pas faire appel à des applications Javascript sophistiquées.
- s'armer de patience car certains codes Javascript élémentaires fonctionnant à merveille sous Netscape, devront être réécrit avec une syntaxe plus stricte.

Pour corser le tout, Microsoft conscient des lacunes de son browser, a sorti une évolution de son interpréteur Javascript. On voit ainsi apparaître fin 97, une JScript.dll version 2. Celle-ci apporte une meilleure compatibilté avec Netscape 3 (comme les objets Array (les tableaux)). Comprendre Explorer 3.0 et Javascript n'était déjà pas simple mais nous voilà maintenant avec du Javascript sous Explorer 3.0 avec la JScript.dll version 1 et la JScript.dll version 2! Quel plaisir ...

Pour Microsoft Explorer 4.0, celui-ci adhère strictement au standard appelé ECMAScript, qui couvre essentiellement la version Javascript 1.1. A fin novembre 97, il est encore un peu tôt pour se faire une idée précise mais il est déjà évident que Microsoft Explorer 4.0 intègre beaucoup mieux (ou très bien ?) le Javascript.

Dans le détail -- et je reprend ce qui suit du site de Danny Goodman (www.dannyg.com) qui est "La" référence en Javascript -- :

Objets non supportés par Explorer 3.0

- L'objet Image[]. Donc pas de onmouseOver sous Explorer 3.
- Pour l'objet Area pas de ommouseOver.
- Pas d'Applet
- Pour les tableaux, écriture stricte Javascript 1.0 pour la version 1. Mais l'objet Array est implanté avec la version 2.
- ...

Propriétés, méthodes, gestionnaires d'événement non supportés

- Window
  onerror closed blur() focus() scroll() onBlur= onFocus=
- Location
  reload() replace()
- Document
  applets[] domain embeds[] images[] URL
- Link
  onMouseOut=
- Form
  reset() onReset=

- (All Form Elements)
  type
- Navigator
  mimeTypes[] plugins[] javaEnabled()
- String
  prototype split()

15.3 Mon script ne fonctionne pas dans un tableau

Javasript dans des tableaux, ce n'est pas une histoire d'amour (bug?). On recommande dans la littérature de ne pas placer de tag <SCRIPT> dans des balises <TD> mais plutôt de commencer le tag <SCRIPT> avant le tag <TD> et d'écrire le tag <TD> jusqu'au tag </TD> par une écriture document.write. Et même, pour être sur de son coup, d'écrire tout la tableau avec le document.write. Ce qui donnerait :

```
<SCRIPT LANGUAGE="Javascript">
<!--
document.write("<TABLE BORDER=1>");
document.write("<TR>");
document.write("<TD>");
document.write("votre texte");
document.write("</TD>");
document.write("<TD>");
document.write("votre texte");
document.write("</TD>");
document.write("</TR>");
document.write("</TABLE>");
//-->
</SCRIPT>
```

En conclusion, au niveau débutant, éviter de mettre du Javascript dans des tableaux. Ajoutons qu'il n'y a pas de problèmes pour les formulaires. Mais est-ce vraiment du Javascript?...

15.3 Adapter le script selon le browser du lecteur

Avec les méthodes et propriétés de l'objet navigator (voir ce chapitre), il y aura moyen de détecter le type et la version du browser. Ce qui sera très utile pour adapter vos scripts au browser et à la version de celui-ci.

La compatibilité des pages Javascript avec les différents types et versions en circulation pose vraiment problème. A l'avenir, on peut espérer une mutation rapide vers Explorer 4 pour les partisans de Microsoft et l'abandon progressif de Netscape 2.0 pour des versions plus récentes. Ce qui simplifiera grandement la situation.
Si vous n'êtes pas obnubilés par la compatiblité de vos pages de script (et les messages d'erreurs qui risquent d'apparaître), soyez quand même sympa d'avertir vos lecteurs. Ainsi, le petit script suivant informe les utilisateurs de Explorer 3.0 qu'ils risquent de rencontrer quelques désagréments.

```
<SCRIPT LANGUAGE = "JavaScript">
<!--
var name = navigator.appName ;
if (name == 'Microsoft Internet Explorer') {
document.write('Attention! Vous utilisez Microsoft Explorer 3.0.') <BR>');
document.write('Avec ce browser, certains scripts peuvent ne pas fonctionner correctement');
}
else { null }
//-->
</SCRIPT>
```

Pour des informations très complètes sur les différentes versions de Javascript, je ne peux que vous conseiller vivement "JavaScript Object Road Map and Compatibility Guide", quatre feuillets que l'on peut télécharger sur le site de Danny Goodman ("La" référence !) à l'adresse www.dannyg.com.

15.4 Arrondir les nombres derrière la virgule

Il arrive que Javascript vous affiche un résultat de division du type 1.599999999999999. Ce qui est assez disgracieux, j'en conviens. Nous approfondirons le sujet lors de l'étude de la méthode Math. Parmi les différents systèmes possibles , je vous propose celui-ci :

variable= Math.round (variable*100)/100
Ainsi, 1.599999 est multiplié par 100 ce qui fait 159.9999. La méthode Math.round (159.9999) donne 160, qui divisé par 100 fait 1.60. Avec ...*100)/100, on obtient 2 chiffres après la virgule. Devinez pour 3 chiffres après la virgule...

15.5 Comment lire et écrire des fichiers en Javascript

Il est impossible de lire un fichier ou d'écrire dans un fichier en Javascript. C'est sûrement très bien pour la sécurité de votre système mais c'est dommage pour, par exemple, l'exploitation d'une base de données.

15.6 Ciel! On voit mes sources Javascript

Et oui, par "View Document Source", le lecteur un peu initié pourra voir, étudier, copier vos sources Javascript. Il existe sur le net plusieurs petits programmes de codage des scripts, d'appels à des fichiers dissimulés, etc. Je suis arrivé à la conclusion que aucun système ne pourra garantir une confidentialité absolue de vos scripts. Si pour les entreprises, cela peut être gênant pour des données sensibles (mais il y a alors d'autres techniques disponibles, je pense à du Java), pour les simples particuliers on peut se demander si cela a vraiment de l'importance... Ne seriez vous pas un peu paranoïaque ? Si oui, il y a toujours moyen de coder votre Javascript d'une façon (presque) incompréhensible pour le programmeur moyen.

15.7 Transmettre des variables d'une page à l'autre

Vos variables sont définies dans le script de l'entité que constitue votre page Web. Vous pouvez souhaiter continuer à utiliser ces variables dans une autre page ou tout au long de votre site. Comment faire? La solution est d'utiliser des frames. Javascript permet de passer des variables d'un frame vers des objets appartenant à un autre frame. Et comme tous les browsers Javascript admettent des frames, pourquoi s'en priver (voir un des chapitres suivant).

15.8 Les boutons radio me renvoient l'ordre inverse

Ce bug propre à Netscape 2, peut vous jouer de grosses surprises par exemple dans un formulaire de commande utilisant des boutons radio. En effet si le bouton radio 1 d'une série de 3 est checked, c'est la valeur 3 qui est retournée. Pour vous, le client a commandé une chemise Large alors qu'il souhaitait du Small !
Pour corriger ce bug, il suffit d'ajouter un gestionnaire d'événement vide à chaque contrôle de la série de boutons radio.

```
<FORM NAME="radioTest">
<INPUT TYPE="radio" NAME="test" VALUE="A" onClick="">A
<INPUT TYPE="radio" NAME="test" VALUE="B" onClick="">B
<INPUT TYPE="radio" NAME="test" VALUE="C" onClick="">C
</FORM>
```

Chapitre 16 : L'objet window

16.1 Propriétés et méthodes de l'objet window

Certaines propriétés et méthodes de l'objet window ne vous sont pas inconnues :
- celles des boites de dialogue. Soit alert(), confirm(), et prompt()
- et celles du ou des minuteries. Soit setTimeout() et clearTimeout().

Une autre série, ayant trait aux frames, fait l'objet d'un chapitre consacré à ce sujet :

- ce sont frames[], length, parent , opener et top.

Une série a trait à la barre d'état qui est abordée ci-après :
- ce sont status et defaultStatus.

Une série pour l'ouverture et la fermeture d'une fenêtre :
- ce sont open() et close().

Et enfin self qui renvoie à la fenêtre en cours.

16.2 Utilisation de la barre d'état

Avec Javascript, la barre d'état (petite bande située au bas de la fenêtre du browser et qui vous informe sur l'état des transferts et des connections) peut être utilisée pour afficher des messages de votre cru. Comme je suis myope comme une taupe, ce n'est pas ma partie préférée du Javascript mais c'est une opinion des plus subjective.

Les propriétés mises en oeuvre sont :

Propriété	Description
status	valeur du texte affiché dans la barre d'état de la fenêtre.
DefaultStatus	valeur par défaut qui s'affiche dans la barre d'état.

Généralement, cet événement est mis en oeuvre par un onmouseOver() sur un lien hypertexte.

En voici un exemple :

```
<HTML>
<BODY>
<A HREF="#" onmouseover="self.status='Votre texte'; return true;"> A voir ici </A>
</BODY>
</HTML>
```

Il est indispensable d'ajouter return true;

16.3 Ouverture et fermeture de fenêtres (théorie)

Les méthodes mises en oeuvre sont :

Méthodes	Description
open()	ouvre une nouvelle fenêtre.
close()	ferme la fenêtre en cours.

La syntaxe est :
[window.]open("URL","nom_de_la_fenêtre","caractéristiques_de_la fenêtre")

où URL est l'URL de la page que l'on désire afficher dans la nouvelle fenêtre. Si on ne désire pas afficher un fichier htm existant, on mettra simplement "".

où caractérstiques_de_la _fenêtre est une liste de certaines ou de toutes les caractéristiques de fenêtre suivantes que l'on note à la suite, séparées par des virgules et sans espaces ni passage à la ligne.

Caractéristique	Description
toolbar=yes ou no	Affichage de la barre d'outils
location=yes ou non	Affichage de champ d'adresse (ou de localisation)
directories=yes ou no	Affichage des boutons d'accès rapide

status=yes ou no	Affichage de la barre d'état
menubar=yes ou no	Affichage de la barre de menus
scrollbars=yes ou no	Affichage des barres de défilement. (scrollbars=no fonctionne mal sous Explorer 3.0)
resizable=yes ou no	Dimensions de la fenêtre modifiables
width=x en pixels	Largeur de la fenêtre en pixels
height=y en pixels	Hauteur de la fenêtre en pixels

On peut aussi utiliser 1 ou 0 au lieu de yes ou no.

Remarques :

Cette nouvelle fenêtre va s'afficher un peu n'importe où sur votre écran. Vous ne pouvez pas décider de l'endroit exact où elle peut apparaître. Cependant sous Netscape 4 c.-à-d. sous Javascript 1.2 , ce petit "plus" est possible.

Sous Microsoft Explorer 3, l'apparition de la nouvelle fenêtre se fait après une grimace du browser (il ouvre temporairement une nouvelle fenêtre du browser).

L'usage des nouvelles fenêtres est assez sympathique en Javascript pour afficher des informations complémentaires sans surcharger la page (ou fenêtre) de départ. Cependant, aussi longtemps que l'utilisateur ne ferme pas ces nouvelles fenêtres, celles-ci restent ouvertes (lapalissade). Le pire est lorsqu'on les minimise. Pour peu qu'on utilise souvent cette technique, le navigateur se retrouve avec plusieurs dizaines de fenêtres ouvertes ce qui fait désordre, ralentit le système et peut finir par le planter.

Veillez donc à toujours faire fermer ces nouvelles fenêtres.

16.4 Ouverture et fermeture de fenêtres (exemples)

--- Ouverture par un bouton (avec code dans le onClick)

Nous allons ouvrir un petite fenêtre qui affiche le fichier test.htm avec un bouton dans la page.

Fichier test.htm :
```
<HTML>
<BODY>
<H1>Ceci est un test<HI>
<FORM>
<INPUT TYPE="button" value= " Continuer " onClick="self.close()">
</FORM>
</BODY>
</HTML>
```
où self.close() fermera la fenêtre courante, c.-à-d. la nouvelle fenêtre.

Dans la page de départ :
```
<FORM>
<INPUT TYPE ="button" value="Ouvrir une nouvelle fenêtre"
onClick="open('test.htm', 'new', 'width=300,height=150,toolbar=no,location=no,
directories=no,status=no,menubar=no,scrollbars=no,resizable=no')">
(sans espaces ni passage à la ligne)
</FORM>
```

--- Ouverture par un bouton (avec appel d'une fonction)

Dans la page de départ :
```
<SCRIPT LANGUAGE="javascript">
<!--
function new_window() {
xyz="open('test.htm', 'new', 'width=300,height=150,toolbar=no,location=no,
```

```
directories=no,status=no,menubar=no,scrollbars=no,resizable=no')">
// sans espaces ni passage à la ligne
}
// -->
</SCRIPT>

<FORM>
<INPUT TYPE ="button" value="Ouvrir une nouvelle fenêtre"
onClick="new_window()">
</FORM>
```

--- Fermeture automatique après x secondes

Avec ce script, sans intervention de l'utilisateur, la nouvelle fenêtre se ferme de façon automatique après 4 secondes. En cliquant sur le bouton, l'utilisateur interrompt prématurément le compteur et ferme la fenêtre. Avec ce système, on est certain que le nouvelle fenêtre sera fermée.

La page test.htm devient testc.htm

```
<HTML>
<BODY onLoad='compt=setTimeout("self.close();",4000)'>
<H1>Ceci est un test</H1>
<FORM>
<INPUT TYPE="button" value=" Continuer " onClick="clearTimeout(compt);self.close();">
</FORM>
</BODY>
</HTML>
```

Dans la page de départ :

```
<FORM>
<INPUT TYPE ="button" value="Ouvrir une nouvelle fenêtre"
onClick="open('testc.htm', 'new', 'width=300,height=150,toolbar=no,location=no,
directories=no,status=no,menubar=no,scrollbars=no,resizable=no')">
(sans espaces ni passage à la ligne)
</FORM>
```

--- Ouverture en cliquant sur un lien ou une image

On ajoute simplement le "onClick=open..." à la balise <A> du lien ou de l'image.

Dans la page de départ, on a :

```
<A HREF="#" onClick="open('testc.htm', '',
'width=300,height=150,toolbar=no,location=no,directories=no,status=no,menubar=
no,scrollbar=no,resizable=no')">lien de test</A> (sans espaces ni passage à la ligne)
```

La fermeture automatique est particulièrement utile ici car si l'utilisateur clique quand même, la nouvelle fenêtre disparaît derrière la fenêtre de départ et risque donc de ne pas être fermée.

--- Ouverture par survol du lien (Javascript 1.0)

On utilise ici onmouseOver. Pour rappel, Javascript 1.0 donc compatible Explorer 3.

Dans la page de départ, on a :

```
<A HREF="#" onmouseOver="open('testc.htm', '',
'width=300,height=150,toolbar=no,location=no,directories=no,status=no,menubar=
no,scrollbar=no,resizable=no')">lien de test</A> (sans espaces ni passage à la ligne)
```

--- Ouverture par survol du lien et fermeture en quittant le lien (Javascript 1.1)

0n utilise ici onmouseOver et onmouseOut. Pour rappel, onmouseOver est du Javascript 1.1 et ne fonctionne donc pas sous Explorer 3.0.

Dans la page de départ, on a :
<A HREF="#" onmouseOver="open('test.htm', '',
'width=300,height=150,toolbar=no,location=no,directories=no,status=no,menubar=
no,scrollbar=no,resizable=no')" onmouseOut="self.close()">lien de test</A>
(sans espaces ni passage à la ligne)

--- Ecrire dans la nouvelle fenêtre

On passe par l'ouverture d'un nouvelle fenêtre par l'appel d'une fonction.

Dans la page de départ :
```
<SCRIPT LANGUAGE="Javascript">
<!--
function opnw(){ msg=window.open("","","width=300,height=50,toolbar=no,location=no,directories=no,
status=no,menubar=no,scrollbars=no,resizable=no");
msg.document.write('<HTML> <BODY>' +
'<CENTER><H1>Ceci est un test<H1></CENTER>' +
'</BODY></HTML>')
// sans espaces ni passage à la ligne
}
// -->
</SCRIPT>
```

Selon l'auteur, avec cette forme d'écriture, il n'y a pas moyen de fermer la fenêtre par un bouton (Help...)

Chapitre 17 : L'objet String

17.1 Généralités

Revenons à l'objet String [Afficher du texte -- Avancé --] pour nous intéresser à la manipulation des caractères si utile pour l'aspect programmation de Javascript.

On signale dans la littérature une limitation de la longueur des strings à 50/80 caractères. Cette limitation du compilateur Javascript peut toujours être contournée par l'emploi de signes + et la concaténation.

Instruction	Description
length	C'est un entier qui indique la longueur de la chaîne de caractères.
charAt()	Méthode qui permet d'accéder à un caractère isolé d'une chaîne.
indexOf()	Méthode qui renvoie la position d'une chaîne partielle à partir d'une position déterminée (en commençant au début de la chaîne pricipale soit en position 0).
LastIndexOf()	Méthode qui renvoie la position d'une chaîne partielle à partir d'une position déterminée (en commençant à la fin soit en position length moins 1).
substring(x,y)	Méthode qui renvoie un string partiel situé entre l position x et la position y-1.
toLowerCase()	Transforme toutes les lettres en minuscules.
toUpperCase()	Transforme toutes les lettres en Majuscules.

17.2 La propriété length

La propriété length retourne un entier qui indique le nombre d'éléments dans une chaîne de caractères. Si la chaîne est vide (" "), le nombre est zéro.

La syntaxe est simple :
x=variable.length;
x=("chaîne de caractères").length;

La propriété length ne sert pas que pour les Strings, mais aussi pour connaître la longueur ou le nombre d'éléments :

- de formulaires . Combien a-t-il de formulaires différents ?
- de boutons radio. Combien a-t-il de boutons radio dans un groupe ?
- de cases à cocher. Combien a-t-il de cases à cocher dans un groupe ?
- d'options. Combien a-t-il d'options dans un Select ?
- de frames. Combien a-t-il de frames "enfants" ?
- d'ancres, de liens, etc.

17.3 La méthode CharAt()

Il faut d'abord bien noter que les caractères sont comptés de gauche à droite et que la position du premier caractère est 0. La position du dernier caractère est donc la longueur (length) de la chaîne de caractère moins 1;

chaîne : `Javascript` (longueur = 10)
`||||||||||`
position : `0123456789` (longueur - 1)

Si la position que vous indiquer est inférieure à zéro ou plus grande que la longueur moins 1, Javascript retourne une chaîne vide.

La syntaxe de charAt() est :
chaîne_réponse = chaîme_départ.charAt(x);

où x est un entier compris entre 0 et la longueur de la chaîne à analyser moins 1.

Notez les exemples suivants :

```
var str="Javascript";                    La réponse est "J".
var chr=str.charAt(0);
var chr="Javascript".charAt(0);
ou var chr=charAt(str,0);
ou var chr=charAt("Javascript",0);
var str="Javascript";                    La réponse est "t".
var chr=str.charAt(9);
var chr=charAt(str,9);
var str="Javascript";                    La réponse est ""
var chr=charAt(str,13);                  soit vide.
```

17.4 La méthode indexOf()

Cette méthode renvoie la position, soit x, d'un string partiel (lettre unique, groupe de lettres ou mot) dans une chaîne de caractères en commençant à la position indiquée par y. Cela vous permet, par exemple, de voir si une lettre, un groupe de lettres ou un mot existe dans une phrase.

variable="chaîne_de_caractères";
var="string_partiel";
x=variable.indexOf(var,y);
où y est la position à partir de laquelle la recherche (de gauche vers la droite) doit commencer. Cela peut être tout entier compris entre 0 et la longueur - 1 de la chaîne de caractères à analyser.
Si y n'est pas spécifié, la recherche commencera par défaut à la position 0.

Si le string partiel n'est pas trouvé dans la chaîne de caractères à analyser, la valeur retournée sera égale à -1.

Quelques exemples :

variable="Javascript"
var="script"
x=variable.indexOf(var,0); x vaut 4variable="VanlanckerLuc&ccim.be"
var="@"
x=variable.indexOf(var); x vaut -1

17.5 La méthode lastIndexOf()

Méthode fort semblable à indexOf() sauf que la recherche va cette fois de droite à gauche (en commençant donc par la fin).

La syntaxe ext en tous points identique sauf que y signale une position située vers la fin de la chaîne de caractères.

x=variable.lastIndexOf(var,y);

Les exemples suivants montrent la différence entre index0f() et lastIndexOf() :

variable="Javascript"
var="a"
x=variable.indexOf(var,0); ici x vaut 1 soit la position du premier a.
x=variable.lastIndexOf(var,9); ici x vaut 3 soit la position du second a.

Notons que même lorsqu'on commence à lire de la fin de la chaîne, la position retournée est comptée depuis le début de la chaîne avec le comptage commençant à zéro.

17.6 La méthode substring()

La méthode substring() est du même genre que indexOf(), lastIndexOf() et charAt() que nous venons d'étudier. Elle sera particulièrement utile, par exemple, pour prendre différentes données dans une longue chaîne de caractères.

variable = "chaîne de caractères"
resultat=variable.substring(x,y)

où resultat est un sous ensemble de la chaîne de caractère (ou de la variable).

Les x et y sont des entiers compris entre 0 et la longueur moins 1 de la chaîne de caractères.
Si x est inférieur à y, la valeur retournée commence à la position x et se termine à la position Y-1.
Si x est supérieur à y, la valeur retournée commence à la position y et se termine à la position X-1.
En fait, ceci donne le même résultat et il est équivalent d'écrire par exemple substring(3,6) ou substring(6,3).

Si x est égal à y, substring() retourne une chaîne vide (logique, non?)

Vous souhaitez sûrement quelques exemples :

```
Javascript
||||||||||
0123456789
```

str="Javascript";
str1=str.substring(0,4);
str2="Javascript".substring(0,4);
str3=str.substring(6,9);

Les résultats sont :
str1="Java"; soit les positions 0,1,2 et 3.
str2="Java"; soit les positions 0,1,2 et 3.

str3="rip"; soit les positions 6,7 et 8

17.7 La méthode toLowerCase()

Cette méthode affiche toutes les majuscules d'une chaîne de caractères variable2 en minuscules.

```
variable2="chaîne de caractères";
variable1=variable2.toLowerCase();
```

Exemple :

```
str="JavaScript";
str1=str.toLowerCase();
str2="JavaScript".toLowerCase();
```

Le résultat sera :

```
str1="javascript";
str2="javascript";
```

17.8 La méthode toUpperCase()

Cette méthode affiche toutes les minuscules d'une chaîne de caractères variable2 en majuscules.

```
variable2="chaîne de caractères";
variable1=variable2.toUpperCase();
```

Exemple :

```
str="JavaScript";
str3=str.toUpperCase();
str4="JavasScript".toUpperCase();
```

Le résultat sera :

```
str3="JAVASCRIPT";
str4="JAVASCRIPT";
```

17.9 Utilité de toLowerCase() et de toUppercase()

L'utilité de ces 2 méthodes ne vous saute peut être pas aux yeux. Et pourtant, il faut se rappeler que Javascript est case sensitive. Ainsi une recherche sur Euro ne donnera pas le même résultat que sur euro ou EUro.

Ainsi, pour les bases de données, il est prudent de tout convertir en minuscules (ou en majuscules). A fortiori, pour certaines informations des utilisateurs introduites par le biais d'un formulaire.

Chapitre 18 : L'objet Math

- Pour manipuler les nombres, voici l'objet Math.

18.1 abs()

```
x=Math.abs(y);
```

La méthode abs() renvoie la valeur absolue (valeur positive) de y. Il supprime en quelque sorte le signe négatif d'un nombre.

```
y = 4;
x = math.abs(y);
x = Math.abs(4);
x = math.abs(-4);
```

ont comme résultat
x = 4

18.2 ceil()

x=Math.ceil(y);

La méthode ceil() renvoie l'entier supérieur ou égal à y.

Attention ! Cette fonction n'arrondit pas le nombre.
Comme montré dans l'exemple, si y = 1.01, la valeur de x sera mise à 2.

y=1.01;
x=Math.ceil(y);
a comme résultat 2.

18.3 floor()

x=Math.floor(y);

La méthode floor() renvoie l'entier inférieur ou égal à y.

Attention ! Cette fonction n'arrondit pas le nombre.
Comme montré dans l'exemple, si y = 1.99, la valeur de x sera mise à 1.

y=1.999;
x=Math.floor(y);
a comme résultat 1.

18.4 round()

x=Math.round(y);

La méthode round() arrondit le nombre à l'entier le plus proche.

y=20.355;
x=Math.round(y);
a comme résultat
x=20;

Attention ! Certains calculs réclament une plus grande précision. Pour avoir deux décimales après la virgule, on utilisera la formule :
x=(Math.round(y*100))/100;
et dans ce cas
x=20.36;

18.5 max()

x=Math.max(y,z);

La méthode max(y,z) renvoie le plus grand des 2 nombres y et z.

y=20; z=10;
x=Math.max(y,z);
a comme résultat
x=20;

18.6 min()

x=Math.min(y,z);

La méthode min(y,z) renvoie le plus petit des 2 nombres y et z.

y=20; z=10;
x=Math.min(y,z);
a comme résultat
x=10;

18.7 pow()

x=Math.pow(y,z);

La méthode pow() calcule la valeur d'un nombre y à la puissance z.

y=2; z=8
x=Math.pow(y,z);
a comme résultat
28 soit 256

18.8 random()

x=Math.random();

La méthode random() renvoie la valeur d'un nombre aléatoire choisi entre 0 et 1.
Attention ! Cela ne fonctionne que sous Unix.
A la place, on peut utiliser la fonction suivante qui renvoie un nombre "aléatoire" entre 0 et 9.
function srand(){ t=new Date(); r=t.getTime(); p="a"+r; p=p.charAt((p.length-4)); x=p; }

18.9 sqrt()

x=Math.sqrt(y);

La méthode sqrt() renvoie la racine carrée de y.

y=25;
x=Math.sqrt(y);
a comme résultat
x=5;

- Voici d'autres conversions

18.10 parseFloat()

x=parseFloat("variable");

Cette fonction convertit une chaîne contenant un nombre en une valeur à virgule flottante. Ou si vous préférez, elle retourne les chiffres derrière la virgule d'un nombre.

Attention ! Le résultat risque d'être surprenant si Javascript rencontre autre chose dans la chaîne que des nombres, les signes + et -, le point décimal ou un exposant. S'il trouve un caractère "étranger", La fonction ne prendra en compte que les caractères avant le caractère "étranger".
Si le premier caractère n'est pas un caractère admis, x sera égal à 0 sous Windows et à "NaN" sur les autres systèmes.

str='-.12345';

str1='$5.50';
x=parseFloat(str);
aura comme résultat
x= -.12345;
x=0 ou "NaN";

18.11 parseInt()

x=parseInt(variable);

Retourne la partie entière d'un nombre avec une virgule.

str='1.2345';
x=parseInt(str);
x=1;

18.12 eval()

x=eval(variable);

Cette fonction évalue une chaîne de caractère sous forme de valeur numérique. On peut stocker dans la chaîne des opérations numériques, des opérations de comparaison, des instructions et même des fonctions.

str='5 + 10';
x=eval(str);
a comme résultat
x=15;

On dit dans la littérature que cette fonction eval() est une opération majeure de Javascript. Que son emploi n'est pas toujours conseillé pour les débutants. Pire, que cette fonction eval() n'est pas supportée par toutes les plate-formes avec toutes les versions des browsers. Prudence donc, vérifiez toujours le résultat renvoyé par eval(). Mieux, trouvez un autre tutorial qui sera plus précis sur le sujet !

- Et pour être complet ou vous assommer complètement

18.13 Les fonctions trigonométriques

Voici (sans commentaires) ces différentes fonctions :

x=Math.PI;
x=Math.sin(y);
x=Math.asin(y);
x=Math.cos(y);
x=Math.acos(y);
x=Math.tan(y);
x=Math.atan(y);

18.14 Les fonctions logarithmiques

Pour les initiés :

x=Math.exp(y);
x=Math.log(y);
x=Math.LN2;
x=Math.LN10;
x=Math.E;
x=Math.LOG2E;
x=Math.LOG10E;

Chapitre 19 : L'objet Date

19.1 new Date();

Cette méthode renvoie toutes les informations "date et heure" de l'ordinateur de l'utilisateur.

variable=new Date();

Ces informations sont enregistrées par Javascript sous le format :
"Fri Dec 17 09:23:30 1998"

Attention ! La date et l'heure dans Javascript commence au 1e janvier 1970. Toute référence à une date antérieure donnera un résultat aléatoire.

La méthode new date () sans arguments renvoie la date et l'heure courante.
Pour introduire une date et une heure déterminée, cela se fera sous la forme suivante :
variable=new Date("Jan 1, 2000 00:00:00");

Toutes les méthodes suivantes vous faciliterons la tâche pour accéder à un point précis de cette variable (en fait un string) et pour modifier si besoin en est le format d'affichage.

19.2 getYear()

variable_date=new Date();
an=variable_date.getYear();

Retourne les deux derniers chiffres de l'année dans variable_date. Soit ici 97.
Comme vous n'avez que deux chiffres, il faudra mettre 19 ou bientôt 20 en préfixe soit
an="19"+variable_date.getYear();

19.3 getMonth()

variable_date=new Date();
mois=variable_date.getMonth();

Retourne le mois dans variable_date sous forme d'un entier compris entre 0 et 11 (0 pour janvier, 1 pour février, 2 pour mars, etc.). Soit ici 11 (le mois moins 1).

<Image>

19.4 getDate();

variable_date=new Date();
jourm=variable_date.getDate();

Retourne le jour du mois dans variable_date sous forme d'un entier compris entre 1 et 31.
Eh oui, ici on commence à 1 au lieu de 0 (pourquoi???).
A ne pas confondre avec getDay() qui retourne le jour de la semaine.

19.5 getDay();

variable_date=new Date();
jours=variable_date.getDay();

Retourne le jour de la semaine dans variable_date sous forme d'un entier compris entre 0 et 6 (0 pour dimanche, 1 pour lundi, 2 pour mardi, etc.).

19.6 getHours();

variable_date=new Date();
hrs=variable_date.getHours();

Retourne l'heure dans variable_date sous forme d'un entier compris entre 0 et 23.

19.7 getMinutes();

variable_date=new Date();
min=variable_date.getMinutes();

Retourne les minutes dans variable_date sous forme d'un entier compris entre 0 et 59.

19.8 getSeconds();

variable_date=new Date();
sec=variable_date.getSeconds();

Retourne les secondes dans variable_date sous forme d'un entier compris entre 0 et 59.

19.9 Exemple 1 : Un script qui donne simplement l'heure.

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="Javascript">
<!--
function getDt(){
dt=new Date();
cal=""+ dt.getDate()+"/"+(dt.getMonth()+1)
+ "/19" +dt1.getYear();
hrs=dt.getHours();
min=dt.getMinutes();
sec=dt.getSeconds();
tm=" "+((hrs<10)?"0":"") +hrs)+":";
tm+=((min<10)?"0":"")+min+":";
tm+=((sec<10)?"0":"")+sec+" ";
document.write(cal+tm);
}
// -->
</SCRIPT>
</HEAD>
<BODY >
<SCRIPT LANGUAGE="Javascript">
<!--
getDt();
// -->
</SCRIPT>
</BODY>
</HTML>
```

19.10 Exemple 2 : Un script avec une trotteuse.

Vous souhaitez peut-être que votre affichage de l'heure change toutes les secondes. Rappeler vous la minuterie setTimeOut [Un peu de tout... Avancé]. Il suffit d'ajouter au script un setTimeOut qui affiche l'heure toutes les secondes. La fonction qui affiche l'heure étant getDt(), l'instruction à ajouter est donc setTimeOut("getDt()",1000); Et le tour est joué.

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="Javascript">
<!--
function getDt(){
dt=new Date();
hrs=dt.getHours();
min=dt.getMinutes();
sec=dt.getSeconds();
tm=" "+((hrs<10)?"0":"") +hrs+":";
tm+=((min<10)?"0":"")+min+":";
tm+=((sec<10)?"0":"")+sec+" ";
document.horloge.display.value=tm;
setTimeout("getDt()",1000);
}
// -->
</SCRIPT>
</HEAD>
<BODY onLoad="getDt()">
<FORM name="horloge">
<INPUT TYPE="text" NAME="display" SIZE=15 VALUE ="">
</FORM>
</BODY>
</HTML>
```

19.11 D'autres propriétés (moins fréquentes peut-être)

variable.getTime();

Retourne l'heure courante dans variable_date sous forme d'un entier représentant le nombre de millisecondes écoulées depuis le 1 janvier 1970 00:00:00.

variable.getTimezoneOffset();

Retourne la différence entre l'heure locale et l'heure GMT (Greenwich, UK Mean Time) sous forme d'un entier représentant le nombre de minutes (et pas en heures).

variable.setYear(x);

Assigne une année à à l'actuelle valeur de variable_date sous forme d'un entier supérieur à 1900.
Exemple : variable_date.setYear(98) pour 1998.

variable.setMonth(x);

Assigne un mois à l'actuelle valeur de variable_date sous forme d'un entier compris entre 0 et 11.
Exemple : variable_date.setMonth(1);

variable.setDate(x);

Assigne un jour du mois à l'actuelle valeur de variable_date sous forme d'un entier compris entre 1 et 31.
Exemple : variable_date.setDate(1);

variable.setHours(x);

Assigne une heure à l'actuelle valeur de variable_date sous forme d'un entier compris entre 1 et 23.
Exemple : variable_date.setHours(1);

variable.setMinutes(x);

Assigne les minutes à l'actuelle valeur de variable_date sous forme d'un entier compris entre 1 et 59.
Exemple : variable_date.setMinutes(1);

variable.setSeconds(x);

Assigne les secondes à l'actuelle valeur de variable_date sous forme d'un entier compris entre 1 et 59.
Exemple : variable_date.setSeconds(0);

variable.setTime(x);

Assigne la date souhaitée dans variable_date sous forme d'un entier représentant le nombre de millisecondes écoulées depuis le 1 janvier 1970 00:00:00. et la date souhaitée.

variable.toGMTString();

Retourne la valeur de l'actuelle valeur de variable_date en heure GMT (Greenwich Mean Time).

variable.toLocaleString()

Retourne la valeur de l'actuelle valeur de variable_date sous forme de String.
Il me semble qu'on aura plus vite fait avec les getHours(), getMinutes() and getSeconds().

Chapitre 20 : L'objet Navigator

20. 1 Généralités

Avec l'objet Navigator, on aura la possibilité d'identifier le browser (ainsi que la version de celui-ci) utilisé par le lecteur. Ce qui en maintes occasions sera très utile sinon indispensable pour assurer la compatibilité de vos pages. Les propriétés sont peu nombreuses mais au combien intéressantes mais parfois un peu obscures.

20.2 navigator.appCodeName

Retourne le nom de code du navigateur.

Cette propriété renvoie toujours "Mozilla". "Mozilla" est un nom de code de Netscape qui l'a utilisé en premier. Microsoft (pour une raison que je vous laisse deviner) l'a également repris. On ne pourra donc pas utiliser cette propriété pour différencier un navigateur de Netscape ou de Microsoft.

document.write("Le code name de votre browser est " +navigator.appCodeName);

20.3 navigator.appName

Retourne le nom ou la marque du browser soit "Netscape", soit "Microsoft Internet Explorer"

Cette propriété sera plus utile pour faire la différence entre la famille Netscape et la famille Microsoft du browser.

document.write("Le nom ou la marque du browser est " +navigator.appName);

20.4 navigator.appVersion

Renvoie des informations consernant la version du navigateur, le système d'exploitation de l'utilisateur, un code de nationalité de la version (avec des variantes).

Cette information prend la forme :

2.0 (Win95; I)
releaseNumber(platform,country)
numéro de la version(système d'exploitation, code nationalité de la version)

Autre exemple du numéro de la série "3.0b5" pour Netscape Navigator 3.0, beta version 5.

Le système d'exploitation est celui sous lequel tourne le browser. "Win 16" pour une version 16 bits de Windows, "Win 95" pour une version 32 bits, "Mac 68" et "MacPPC" pour les Macintosh et Power PC (sous Netscape).

Le code de nationalité de la version peut-être " I " pour une version internationale ou "U" pour les versions proprement américaines (sous Netscape).

document.write("Les informations sur la version sont "+navigator.appVersion);

20.5 navigator.userAgent

Renvoie également des informations (sur le header envoyé dans le protocole HTTPd du server de votre visiteur).

document.write("Le browser a comme user-agent name "+navigator.userAgent);

20.6 Passage en revue des troupes

Le petit script, appliqué aux browsers usuels (sous Windows), donne les résultats suivants :

```
<HTML>
<BODY>
<SCRIPT LANGUAGE="javascript">
document.write("Le code name de votre browser est " +navigator.appCodeName);
document.write("Le nom ou la marque du browser est " +navigator.appName);
document.write("Les informations sur la version sont "+navigator.appVersion);
document.write("Le browser a comme user-agent name "+navigator.userAgent);
</SCRIPT>
</BODY>
<HTML>
```

Il est judicieux de remarquer que la longueur du string retourné varie d'un browser à l'autre.

- Netscape 2.0

Le code name de votre browser est Mozilla
Le nom ou la marque du browser est Netscape
Les informations sur la version sont 2.0 (Win95; I)
Le browser a comme user-agent name Mozilla/2.0 (Win95; I)

- Netscape 3.0

Le code name de votre browser est Mozilla
Le nom ou la marque du browser est Netscape
Les informations sur la version sont 3.0Gold (Win95; U)
Le browser a comme user-agent name Mozilla/3.0Gold (Win95; U)

- Netscape 4.0 (Communicator)

Le code name de votre browser est Mozilla
Le nom ou la marque du browser est Netscape
Les informations sur la version sont 4.01 [en] (Win95; I)

Le browser a comme user-agent name Mozilla/4.01 [en] (Win95; I)

- Microsoft Explorer 3.0

Le code name de votre browser est Mozilla
Le nom ou la marque du browser est Microsoft Internet Explorer
Les informations sur la version sont 2.0 (compatible; MSIE 3.0A; Windows 95)
Le browser a comme user-agent name Mozilla/2.0 (compatible; MSIE 3.0A; Windows 95)

- Microsoft Explorer 4.0

Le code name de votre browser est Mozilla
Le nom ou la marque du browser est Microsoft Internet Explorer
Les informations sur la version sont 4.0 (compatible; MSIE 4.0; Windows 95)
Le browser a comme user-agent name Mozilla/4.0 (compatible; MSIE 4.0; Windows 95)

20.7 Exemples

Les scripts pour connaître le browser de l'utilisateur sont nombreux et dépendent un peu de l'inspiration de l'auteur. En voici quelques uns que nous détaillons.

Vous remarquerez que l'on utilise souvent indexOf() car la longueur du string retourné varie d'une version à l'autre.

- Pour des instructions sous Netscape 3 ou 4

```
var nm=navigator.appName+navigator.appVersion;
if(nm.indexOf("Netscape3.")>-1||nm.indexOf("Netscape4.")>-1) ...
```

Ici l'auteur assemble dans la variable nm les informations retournées par appName et appVersion. Avec indexOf, on voit si Netscape3 ou Netscape4 est inclus dans nm. Si oui, sa position sera 0 ou supérieur. Si la référence n'est pas trouvée, la valeur retournée sera -1.

- Pour savoir si le browser tourne sous Windows

On peut employer if (navigator.appVersion.IndexOf('Win')>-1) { ...
Avec indexOf, on teste la position du string partiel Win dans le string retourné par AppVersion. Si Win est trouvé cette position sera notée de 0 à une position déterminée et donc supérieure à -1. Si Win n'est pas trouvé la position retournée sera de -1 et la condition ne sera pas remplie.

- Pour savoir si le browser est Explorer 4.0

```
var ms = navigator.appVersion.indexOf("MSIE")
ie4 = (ms>0) && (parseInt(navigator.appVersion.substring(ms+5, ms+6)) >= 4)
```

Avec indexOf, on note sous la variable ms, la position de MSIE dans le string retourné par app.Version. Si MSIE n'est pas trouvé (donc Netscape), ms vaudra -1.
On donne à ie4 la valeur true, si d'abord ms est supérieur à 0 (de la famille Microsoft) et si le caractère 4 ou supérieur est bien trouvé en position ms+5 du string retourné par app.version.

- Pour distinguer du Javascript 1.0 et du 1.1

Seuls Netscape 2.0 et Explorer 3.0 supportent le Javascript commun, appelé depuis Javascript 1.0. Seuls ces deux browsers ont dans leur userAgent, le chiffre 2.0 (vou pouvez vérifier...).
Ainsi, un test pour déterminer si 2. est bien présent dans le string retourné par userAgent fera l'affaire. Le test devient :

```
var test=navigator.useAgent;
if(test.indesOf("2.") != -1) {...
```

Chapitre 21 : L'objet Array

21.1 Généralités

L'objet Array (ou tableaux) est une liste d'éléments indexés dans lesquels on pourra ranger (écrire) des données ou aller reprendre ces données (lire).

Attention ! *L'objet Array est du Javascript 1.1*

21.2 Tableau à une dimension

Pour faire un tableau, il faut procéder en deux étapes :
- d'abord construire la structure du tableau. A ce stade, les éléments du tableau sont vides.
- ensuite affecter des valeurs dans les cases ainsi définies.

On commence par définir le tableau :

```
nom_du_tableau = new Array (x);
```

où x est le nombre d'élément du tableau.

On notera que, "*le nombre d'éléments est limité à 255. Cette restriction ne figure pas dans la documentation de Netscape mais elle a été constatée expérimentalement*." Source : Javascript de Michel Dreyfus Collection Mégapoche.

Ensuite, on va alimenter la structure ainsi définie :

```
nom_du_tableau[i] = "elément";
```

où i est un nombre compris entre 0 et x moins 1.

Exemple : un carnet d'adresse avec 3 personnes

construction du tableau :

```
carnet = new Array(3);
```

ajout des données :

```
carnet[0]="Dupont";
carnet[1]="Médard";
carnet[2]="Van Lancker";
```

pour accéder un élément, on emploie :

```
document.write(carnet[2]);
```

On notera ici que les données sont bien visibles au lecteur un peu initié (view source).

Remarques :
- Quand on en aura l'occasion, il sera pratique de charger le tableau avec une boucle. Supposons que l'on ait à charger 50 images. Soit on le fait manuellement, il faut charger 0.gif, 1.gif, 2.gif... Soit on utilise une boucle du style :
  ```
  function gifs() {
  gif = new Array(50);
  for (var=i;i<50;i++)
  {gif[i] =i+".gif";}
  }
  ```
- Javascript étant un langage peu typé, il n'est pas nécessaire de déclarer le nombre d'élément du tableau (soit x). Javascript prendra comme nombre d'éléments, le nombre i le plus élevé lors de "l'alimentation" de la structure (en fait i + 1).Ainsi la formulation suivante serait aussi correcte pour un tableau à 3 dimensions.
  ```
  carnet = new Array();
  carnet[2]="Van Lancker";
  ```

21.3 Propriétés et méthodes

Elément	Description
length	Retourne le nombre d'éléments du tableau.
join()	Regroupe tous les éléments du tableau dans une seule chaîne. Les différents éléments sont séparés par une caractère séparateur spécifié en argument. Par défaut, ce séparateur est une virgule.
reverse()	Inverse l'ordre des éléments (ne les trie pas).
sort()	Retourne les éléments par ordre alphabétique (à condition qu'ils soient de même nature)

Dans le cadre de notre exemple,

document.write(carnet.join()); donne comme résultat : Médard,Dupont,Van Lancker.
document.write(carnet.join("-")); a comme résultat : Médard-Dupont-Van Lancker.
document.write(carnet.reverse().join("-")); donne : Van Lancker-Dupont-Médard

21.4 Tableau à deux dimensions

On peut créer des tableaux à deux dimensions (et plus encore) par un petit artifice de programmation.

On déclare d'abord un tableau à 1 dimension de façon classique :

```
nom_du_tableau = new Array (x);
```

Ensuite, on déclare chaque élément du tableau comme un tableau à 1 dimension :

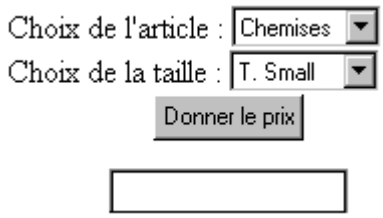
```
nom_du_tableau[i] = new Array(y);
```

Pour un tableau de 3 sur 3 :

Tarif	T. Small	T. Médium	T. Large
Chemises	1200	1250	1300
Polos	800	850	900
T-shirts	500	520	540

```
nom = new Array(3);
nom[0] = new Array(3);
nom[1] = new Array(3);
nom[2] = new Array(3);
nom[0][0]="1200"; nom[0][1]="1250"; nom[0][2]="1300";
nom[1][0]="800"; nom[1][1]="850"; nom[1][2]="900";
nom[2][0]="500"; nom[2][1]="520"; nom[2][2]="540";
```

Pour exploiter ces données, voici une illustration de ce qui est possible :



Le formulaire s'écrit :

```
<FORM name="form" >
<SELECT NAME="liste">
<OPTION>Chemises
<OPTION>Polos
<OPTION>T-shirts
</SELECT>
<SELECT NAME="taille">
<OPTION>T. Small
<OPTION>T. Médium
<OPTION>T. Large
</SELECT>
<INPUT TYPE="button" VALUE="Donner le prix" onClick="affi(this.form)">
<INPUT TYPE="TEXT" NAME="txt">
</FORM>
```

où la fonction affi() se formule comme suit :

```
function affi() {
i = document.form.liste.selectedIndex;
j= document.form.taille.selectedIndex
document.form.txt.value=nom[i][j];
}
```

21.5 Base de données

Que voilà un titre ronflant ! Si Javascript peut stocker des données et présenter celles-ci selon les souhaits du client, on est quand même très loin des outils spécifiques pour ce genre de travail. Vos bases de données en Javascript seront toujours de type statique et surtout d'un encodage laborieux.

Jusqu'à présent, nous n'avions défini que quelques caractères dans les éléments des tableaux. Dans les strings, on peut en mettre beaucoup et surtout avec les outils de manipulation des strings de Javascript, on pourra stocker des choses du genre :
Van Lancker*Luc*Rue des brasseurs, 22*7700*Mouscron*Belgium*
Soit dans le cas présent, 6 données.

21.5.1 1Encodage de type fixe

Reprenons notre exemple de carnet d'adresse. De façon classique, on doit prévoir les différents champs et le nombre de positions consacrées à chaque champ. Soit par exemple :

Nom	20 positions
Prénom	10 positions
Adresse	20 positions
Code postal	10 positions
Ville	10 positions
Pays	10 positions

Chaque enregistrement devra respecter ce nombre de positions prédéterminées. Les positions surnuméraires devront être complétées avec des blancs. Avec un risque d'erreur élevé à l'encodage et un temps de chargement de la page sensiblement augmenté pour finalement ne transmettre que des espaces vides.

Nom	.Prénom	Adresse	Poste	Ville	Pays
sur 20 p.	sur 10 p.	sur 20 p.	sur 10 p.	sur 10 p.	sur 10 p
Van Lancker	Luc	Rue des Brasseurs	7700	Mouscron	Belgium

Pour reprendre les différentes données, on utilisera l'instruction substring(x,y). Ainsi :

Nom	substring(0,19)
Prénom	substring(20,29)

Adresse	substring(30,49)
Code postal	substring(50,59)
Ville	substring(60,69)
Pays	substring(70,79)

21.5.2 Encodage de type variable

Le principe est d'ajouter un séparateur entre chaque champ. On demandera à Javascript de retirer les caractères compris entre deux séparateurs.

Avec le séparateur *, les données deviennent :
str="Van Lancker*Luc*Rue des brasseurs, 22*7700*Mouscron*Belgium*"
soit 60 positions au lieu de 80 soit un gain de 25 % (mais l'échantillon n'est pas représentatif).

Pour lire ces différentes données, on va procéder en plusieurs étapes :

- Par pos=str.indexOf("*"), on note la position dans le string du premier séparateur rencontré.
- Par str.substring(0,pos), on aura la première donnée comprise entre le début du string (position 0) et la position moins 1 du séparateur soit ici Van Lancker.
- Par str=str.substring(pos+1,str.length), on recrée un string correspondant au string de départ moins la donnée partielle que l'on vient d'extraire moins un séparateur. Ce qui donne :
  str="Luc*Rue des brasseurs, 22*7700*Mouscron*Belgium*"
- Et ceci, par une boucle, autant de fois qu'il y a de séparateurs mois 1.

Pour stocker des données, le moyen le plus pratique pour stocker des données est le contrôle de formulaire Hidden (caché) plutôt que le string.

Pourquoi, me direz vous ? Bien qu'avec des string cela fonctionne, on risque de buter sur la limitation de longueur des strings en Javascript de, selon la littérature, 50 à 80 caractères. Ce qui entraîne l'emploi de signes + et de la concatenation des données pour qu'elles soient acceptées par le compilateur Javascript.

Le contrôle de formulaire Hidden n'est quand à lui pas sujet à cette limitation de caractères. On peut y stocker 30 K (et plus) de données. Il suffira alors d'entrer dans de contrôle de formulaire Hidden l'attribut value="les données à stocker".

Le script complet sera quelque chose du genre :

```
<HTML>
<BODY>
<FORM name="form">
<INPUT type=hidden name="data" value="Van Lancker*Luc*Rue des brasseurs,
22*7700*Mouscron*Belgium*">
</FORM>
<SCRIPT LANGUAGE="javascript">
str=document.form.data.value; // on extrait le string du contrôle caché
nsep=6 // nombre de séparateurs
for (var i=0;i<nsep;i++){
pos=str.indexOf("*");
document.write(str.substring(0,pos)+ "<BR>");
str=str.substring(pos+1,str.length);
}
</SCRIPT>
</BODY>
</HTML>
```

Chapitre 22 : Les frames

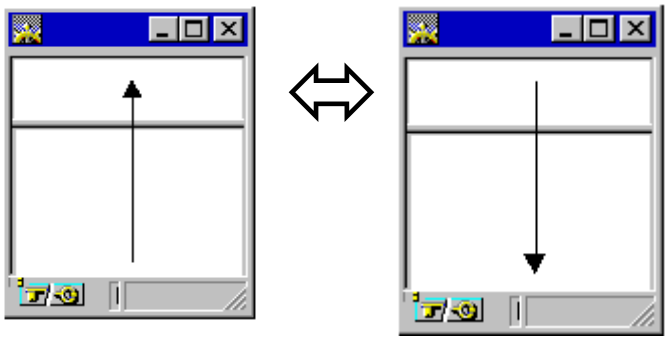
22.1 Généralités

Tout d'abord, présentons les frames. Utiliser des frames vous permet de diviser la fenêtre affichant votre page HTML en plusieurs cadres (parties distinctes) indépendants les uns des autres. Vous pouvez alors charger des pages différentes dans chaque cadre. Pour la syntaxe Html des frames, vous pouvez vous référer au tutorial du même auteur "Apprendre le langage Html - Les frames -" à l'adresse www.ccim.be/ccim328/html/index.htm

En Javascript, nous nous intéresserons à la capacité de ces cadres à interagir. C'est à dire à la capacité d'échanger des informations entre les frames.

En effet, la philosophie du Html veut que chaque page composant un site soit une entité indépendante. Comment pourrait-on faire alors pour garder des informations tout au long du site ou tout simplement passer des informations d'une page à une autre page. Tout simplement (sic), en utilisant des frames.

Le schéma suivant éclairera le propos :



22.2 Propriétés

Propriétés	Description
length	Retourne le nombre de frames subordonnés dans un cadre "créateur".
parent	Synonyme pour le frame "créateur".

22.3 Echange d'informations entre frames

Par l'exemple suivant, nous allons transférer une donnée introduite par l'utilisateur dans une frame, dans une autre frame.

La page "créatrice" des frames

```
<HTML>
<HEAD>
</HEAD>
<FRAMESET ROWS="30%,70%">
<FRAME SRC="enfant1.htm" name="enfant1">
<FRAME SRC="enfant2.htm" name="enfant2">
</FRAMESET>
</HTML>
```

La page "créatrice" contient deux frames subordonnées "enfant1" et "enfant2".

Le fichier enfant1.htm

```
<HTML>
<BODY>
<FORM name="form1">
<INPUT TYPE="TEXT" NAME="en" value=" ">
```

```
</FORM>
</BODY>
</HTML>
```

Le fichier enfant2.htm

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="Javascript">
<!--
function affi(form) {
parent.enfant1.document.form1.en.value=document.form2.out.value
}
// -->
</SCRIPT>
</HEAD>
<BODY>
Entrez une valeur et cliquez sur "Envoyer".
<FORM NAME="form2" >
<INPUT TYPE="TEXT" NAME="out">
<INPUT TYPE="button" VALUE="Envoyer" onClick="affi(this.form)">
</FORM>
</BODY>
</HTML>
```

La donnée entrée par l'utilisateur se trouve par le chemin document.form2.out.value. Nous allons transférer cette donnée dans la zone de texte de l'autre frame. Pour cela, il faut en spécifier le chemin complet. D'abord la zone de texte se trouve dans la frame subordonnée appelée enfant1. Donc le chemin commence par parent.enfant1. Dans cette frame se trouve un document qui contient un formulaire (form1) qui contient une zone de texte (en) qui a comme propriété value. Ce qui fait comme chemin document.form1.en.value. L'expression complète est bien :
parent.enfant1.document.form1.en.value=document.form2.out.value

---

Auteur :
Van Lancker Luc
Rue des Brasseurs, 22
7700 Mouscron Belgium
Vanlancker.Luc@ccim.be

Un mot d'encouragement ou un compliment fait toujours plaisir.
Critiques et suggestions seront aussi examinées avec attention.

Du même auteur :

Apprendre le langage Html	www.ccim.be/ccim328/html/index.htm
Maîtriser le langage Html	www.ccim.be/ccim328/htmlplus/index.htm
Apprendre le Vbscript	www.ccim.be/ccim328/vb/index.htm
Apprendre à créer un site	www.ccim.be/ccim328/site/index.htm